Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Машинско инжењерство

Ниво студија: Основне академске студије

Модул: Информатика у инжењерству

Предмет: Софтверски инжењеринг

Број индекса: 708/2013

Љубомир Д. Живановић

Развој софтверске апликације за израчунавање раздаљине између било која два места у свету

завршни рад

Комисија за преглед и одбрану:

1. **Др Ненад Филиповић, проф. - ментор**
2. _____
3. _____

Датум одбране:_____

Оцена:_____

Име кандидата:	Љубомир Д. Живановић
Број индекса:	708/2013
Студијски програм:	Машинско инжењерство
Ниво студија:	Основне академске студије
Модул:	Информатика у инжењерству
Предмет:	Софтверски инжењеринг
Назив завршног рада:	Развој софтверске апликације за израчунавање раздаљине између било која два места у свету
Ментор:	Др Ненад Филиповић, проф.

У оквиру овог завршног рада кандидат треба да

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Препоручена литература:

[1] .....

[2] .....

[3] .....

Крагујевац, датум

ментор:

Др Ненад Филиповић, проф.

_____

Резиме рада: У овом раду је приказан развој задате апликације коришћењем Visual Studio 2008 развојног окружења и програмског језика C#. Представљен је поступак креирања апликације која у себи треба да поседује могућност повезивања градова у целом свету и исписује информације о њима, уз коришћење Google map-a. Рад је употпуњен графичким приказом корака од почетка израде, па све до стартовања и тестирања апликације.

Кључне речи: Visual Studio 2008, C#, Објектно Оријентисано Програмирање, .NET Framework, Gmap, Windows, променљиве, методе, форме, Radio Button, Timer, Text Box.

Summary: This paper presents the development of the given application using Visual Studio 2008 development environment and programming language C #. Presented is a method of creating applications which in itself should have the ability to connect cities all over the world and prints the information about them using a Google map. The work is complemented by a graphical representation of the steps from the beginning of creation, to the start-up and testing application.

Keywords: Visual Studio 2008, C#, Object-Oriented Programming, .NET Framework, Gmap, Windows, variables, methods, forms, Radio Button, Timer, Text Box.

САДРЖАЈ

УВОД

Основно средство комуникације између учесника у процесу комуникације јесте језик. Програмски језици омогућавају креирање апликативног софтвера. Они омогућавају корисницима да саопште рачунарима шта треба да ураде и представљају основу за развој система. Програмски језици представљају скуп симбола и правила за писање програмског кода.

Програмирање је начин управљања помоћу дефинисања програма рада. За решавање комплексних математичких проблема постоје веома развијени програмски језици. Ови програми омогућују руковање елементарним операцијама, скуповима унапред програмираних операција.

Сваки језик користи другачији скуп правила и синтаксу која одређује правило за грађење одређене наредбе. Успех комуникације између учесника у процесу комуникације зависи од језика који мора бити разумљив. Начин комуникације између човека и компјутера је специфичан, тако да директна комуникација између човека и компјутера није могућа. Језик компјутера састоји се из симбола (0 и 1), а језик човека је богат и разноврстан.

Таквим језиком је било потребно одрадити апликацију која ће повезати било која два града у свету и приказати информације о њима. У времену где компјутери постају свакодневница, и где је непојмљиво више било шта радити без икакве помоћи рачунара, овакав вид апликације није новост. Овако нешто је већ развијено, али у наредном раду ће бити приказана манипулација унапред одређеним садржајем, и ауторов лични печат што се тиче развоја поменуте апликације у развојном окружењу Visual Studio 2008, и његовој алатки C#.

1
ПРОГРАМСКИ ЈЕЗИЦИ

Програмски језик је вештачки језик који се може користити за контролу понашања машине, нарочито рачунара. Програмски језици су дефинисани преко синтаксних и семантичких правила која респективно описују њихову структуру и значење. Многи програмски језици имају неку форму писаних спецификација њихове синтаксе и семантике а неки су дефинисани једино преко званичне имплементације. Програмски језици се користе да олакшају комуникацију са рачунаром приликом организовања и манипулације информација, али и да прецизно изразе алгоритме. Неки аутори ограничавају израз „програмски језик“ само на језике којима се могу изразити сви могући алгоритми, а понекад се користи израз „рачунарски језик“ који се односи на више ограничене вештачке језике. У међувремену је створено више хиљада програмских језика, и нови се стварају сваке године.

Програмски језик може бити било који од вештачких језика којим је могуће дати детаљне инструкције рачунару. Те инструкције се могу извршавати директно када су уграђене у рачунар у посебном облику који је одредио произвођач, тзв. машински језик, после једноставног процеса замене изражене у одговарајућем асемблерском језику, или после превођења из неког језика вишег нивоа.

Машински и асемблерски језици су језици ниског нивоа, који захтевају од програмера да се посвети управљању свим стварима везаним за чување података и операције над њима. На другом крају налазе се језици високог нивоа, који су ближи природном језику и ослобађају програмера бриге о тим стварима, такође читљивији и далеко лакши за писање програма.

Програмски језици се, према начину описивања рада програма, деле на функцијске (Lisp, Skim), процедуларне (C, Paskal, Basic), секвенцијалне и објектно-оријентисане (Java, Ada), структуралне (SQL) и многе друге. Програмски језици по овој подели могу бити мешовити, тј. да дозвољавају различите парадигме у оквиру истог програма, те нпр. C++ дозвољава и објектно-оријентисани и процедурални приступ, штавише процедурални приступ је неопходана при дефиницији почетне тачке програма у функцији main.

Машински језик се састоји од нумеричког кода за операције који одређени рачунар може директно извршити. Тај код је алфанумеричка серија 0 и 1, или бинарни код (бајт), који се често претвара у хексадецимални код (на бази броја 16), ради лакше читљивости и модификације. Инструкције машинских језика обично користе један број бајтова за представљање операција, сабирање на пример, а други за представљање операнда (бројева са којима се врши операција) и/или локације за следећу инструкцију. Асемблерски језик је један ниво изнад машинског језика. Користи кратки мнемонички код за инструкције и омогућава

програмеру да уноси имена за блокове меморије која садржи податке. Дизајниран је да омогући лако превођење у машински језик. Иако се блокови података у асемблерском језику позивају преко имена, а не преко адресе у меморији, ипак не постоји могућност софистикованог организовања сложених информација. Као и машински језик, асемблерски језик подразумева да програмер има одређено знање које ће му омогућити детаљно разрађивање рачунарске архитектуре. Корисно је пре свега због свих тих детаља који су важни, тј. приликом програмирања одређеног рачунара за међусобну интерацију са другим улазним и излазним уређајима, као нпр. штампач, скенер, одређени уређаји за складиштење информација и важних података и друге пре свега оптичке и чврсте дискове. Помоћу њих се могу изражавати алгебарске операције на исти начин као и у математици, и где омогућавају коришћење потпрограма у којима се пакују најчешће коришћене операције, где се омогућава и њихово поновно искоришћавање. Машински језик као језик је тежак за писање и читање, јер не личи на уобичајено математичко представљање нити личи на природни језик, а његов сопствени код варира од рачунара до рачунара.

1.1 Објектно-Оријентисано Програмирање

Објектно оријентисано програмирање је може се слободно рећи парадигма програмирања, јер користи објекте као основу за одређено пројектовање појединих рачунарских програма и многих различитих апликација софтвера. Такође треба нагласити да се заснива на различитим техникама као што су наслеђивање, полиморфизам, енкапсулација, и модуларност.

Решавање проблема парадигмом објектног-оријентисаног програмирања је може се рећи исто као и начин на који људи размишљају и решавају одређене проблеме. Састоји се пре свега од индентификације објекта, а након тога и постављања објекта, који ће се користити, у одговарајућу секвенцу за решење одређеног проблема. Ту се пре свега ради о дизајну одређеног облика чија ће понашања као јединица у одређеној међусобној интеракцији решити одређени проблем. Та интеракција између објеката се састоји у једној размени порука, где свака одређена порука покреће енкапсулиране операције у поменутом објекту и самим тим решава део обично већег и ширег, али и сложенијег проблема. Гледано са аспекта решавања таквих проблема, објектно-оријентисано програмирање то обавља у четири корака:

1. индентификовање проблема
2. индентификовање објеката који су потребни за његово решење
3. индентификовање порука које ће објекти међусобно слати и примати
4. креирање секвенце порука објектима, које ће решавати проблем или проблеме.

У парадигми објектно оријентисаног програмирања, сами објекти су структуре података које представљају одређено и добро, тј. јасно дефинисано знање о спољашњем свету, тј стварности. Класична организација је у хијерархијској класи, причему свака класа објекта поседује одређене информације о особинама објекта које се чувају у инстанцама променљивих, где су све

повезане (концептом асоцијације) са сваком инстанцом понаособ у одређеној класи. Сваки објекат у стању је да може да препозна други објекат, само преко његовог интерфејса. Подаци и сва остала логика сваког објекта су скривени од других објеката, јер се тиме омогућава раздвајање имплементације од одређеног понашања објекта у интеракцији са осталим објектима. Основне особине објекта су: идентитет, стање и понашање. Идентитет представља назив објекта којим се одређени објекат разликује од свих осталих. Понашање објекта је дефинисано операцијама које је могуће извршити над објектом, а активирање одређене операције се извршава поруком. И стање објекта, које је део прошлости и садашњости и које одређује понашање објекта у будућности.

Свака класа се састоји од:

- података-чланова
- објекат-члан
- функција чланица-метода

Податак члан и објекат члан су атрибути класе, и користећи њих описујемо стање објеката. Објекат је модел ентитета где су атрибути есенцијалне особине ентитета које га описују. Методе дефинишу понашање објеката. Темељи објектно-оријентисаног програмирања су:

- Апстракција
- Енкапсулација
- Модуларност
- Полиморфизам
- Наслеђивање

Апстракција је поступак раздвајања битног од небитног. У току овог поступка неопходно је уочити који подаци и везе су битне за дати домен проблема, и оне битне податке имплементирати у класи. Постоје три групе апстракције: апстракција предмета, апстракција процеса и синтетска апстракција (апстракција виртуелне машине).

Енкапсулација подразумева поступке којима се обједињавају стања и понашања у једну целину. Излазни резултат свега овога је класа. Поред тога, енкапсулација такође подразумева и обезбеђивање контроле приступа у циљу поштовања принципа скривања информација.

Модуларност Модуларност је поступак разбијања програма на мање делове, где се добија могућност да они могу сами, и аутономно да функционишу. Модуларност се примењује у циљу омогућења вишеструке употребе софтверских компоненти, односно тзв. поновна употреба кода (code reuse).

Полиморфизам У директној вези са наслеђивањем је и полиморфизам, који представља вероватно једно од моћнијих својстава објектно оријентисаног програмирања. То подразумева да објекат извршава операцију на начин својствен изведеној класи којој припада, иако му се приступа као објекту основне класе.

Наслеђивање Један од фундаменталних аспеката ООР-а је наслеђивање. Што се тиче наслеђивања, ту постоји више врста, као што је наслеђивање имплементације, наслеђивање интерфејса, итд. Такође поред тога, зависно од

програмског језика и његове ООР имплементације, наслеђивање се може поделити на једноструко и вишеструко. Суштинска дефиниција наслеђивања поред свега је то да оно представља везу између класа у којој једна класа наслеђује структуру и понашање друге класе-једноструко наслеђивање, или више класа-вишеструко наслеђивање. У наредном случају се ради о програмском језику C#, који подржава само једноструко наслеђивање. Класа која је наслеђује назива се базном или основном класом, а класа која је наслеђује, назива се изведеном класом. За наслеђивање се може рећи да ке хијерархијска организација класа, где једна класа наслеђује другу и задржава комплетан садржај класе коју наслеђује, а тај садржај може проширити, или редефинисати.

1.2 Visual Studio 2008 развојно окружење

Visual Studio 2008 је јединствен скуп алатки које омогућава програмерима и развојним тимовима да направе квалитетне апликације на Microsoft-овим платформама. Подршка за развој оперативног система Windows Vista и система Microsoft Office помаже програмерима да креирају ефектне и богате клијентске апликације. Сада, уз укључену подршку ASP.NET AJAX и програмски додатак Silverlight за Visual Studio 2008, програмери могу такође да граде велики број богатих интерактивних апликација за Web.

VS представља једну од моћнијих програмских алатки-тј визуелно развојно окружење које омогућава лако коришћење многих технолошких погодности које пружа .NET Framework и C#. Веома просто руковање у ствари подразумева да VS у прилично великом делу помаже кориснику, и мноштво других сложених послова обавља уместо њега. Преведено то подразумева да уместо корисника прави целокупан костур апликације и тиме ослобађа корисника од тог сложеног посла и учења самог начина на који би то морао да ради. Како се Microsoft платформа буде и даље развијала у својим могућностима кроз производе система Windows Server 2008 и SQL Server 2008, Visual Studio 2008 ће наставити да буде јединствено окружење које је програмерима и развојним тимовима неопходно за успешан рад. Прво издање програма Visual Studio појавило се на тржишту 1997 године и садржао је одвојене IDE-ове (који су захтевали посебну инсталацију) за програме Visual C++, Visual Basic, J++, и алатку познату као InterDev. Програм Visual Studio 6.0 имао је драстична побољшања која су означила рођење програма Visual Basic 6 и која су осликавала идеју скупа јединствених услуга за све језике. Visual Studio .NET 2002 i Visual Studio .NET 2003 су ту идеју спровели у дело са апликацијом .NET Framework. По први пут један програмер је могао да напише апликацију на језику по свом избору коришћењем могућности општег скупа алатки укључујући дизајнере, контроле „превуци и отпусти“ и IntelliSense. Заједно са порастом продуктивности програмера, дошло је и до пораста у величини и комплексности развојних пројеката и тимова. Visual Studio 2005 је настао да би помогао програмерима у тимовима различитих величина да повећају међусобну сарадњу и смање комплексност развоја. Са сваким новим издањем, Microsoft је поново потврдио своју посвећеност оспособљавању програмера тако што је оставарио комуникацију са заједницом како би дала повратне информације ради усавршавања производа. Ни Visual Studio 2008 није изузетак. Фокус сваког новог

издања програма Visual Studio увек је настојао да буде што бољи и успешнији. У Visual Studio 2008 продуктивност програмера се не завршава са уређивачем кода и чаробњацима. Проширивањем фокуса на продуктивност, архитектуре за апликације и основну платформу, Visual Studio 2008 вам омогућава да решавате нове пословне проблеме док истовремено смањујете укупне прошкове креирања апликација. [1]

1.3 C# (C Sharp)

C# је једноставан објектно-оријентисан програмски језик опште намене. Развијен је од стране Microsoft тима, који је водио Андрес Хејлсберг. Прва верзија (C# 1.0) се појавила 2001 године, па су се убрзо појављивале нове верзије овог програмског језика- Последња верзија C# је 4.0, која је завршена 12 Априла 2010 године. Пошто је C# наследник C и C++ језика, добио је име sharp по инспирацији музичке нотације што значи да се написана нота изводи за пола корака више.Фајлови писани у овом језику имају екстензију cs. Поменути језик је настао као саставни део Мајкрософтовог развојног окружења .NET Framework 1.0, па је самим тим и прихваћен као језик опште примене и намењен је пре свега за израду апликација за .NET Framework платформу.

Најважније особине језика:

- Не постоји вишеструко наслеђивање, тј. класа може бити потомак само једне класе
- Не постоје глобалне применљиве нити методе. Све променљиве и функције морају бити декларисане унутар класа
- Из разлога енкапсулације постоји образац којим атрибутима различитих класа можемо споља приступити индиректно и то методама get и set.
- Постоје разне конверзије (имплицитне и експлицитне) између различитих типова података (32-битни у 64-битне бројеве, int у float или string...).
- Језик је Case sensitive, разликује мала и велика слова.

Кључне речи C# су:
abstract, as, base, bool, break, byte, case, catch, char, checked, class, const, continue, decimal, default, delegate, do, double, else, enum, event, explicit, extern, false, finally, fixed, float, for, foreach, goto, if, implicit, in, int, interface, internal, is, lock, long, namespace, new, null, object,
operator, out, override, params, private, protected, public, readonly, ref, return, sbyte, sealed, short, sizeof, stackalloc, static, string, struct, switch, this, throw, true, try typeof, uint, ulong unchecked unsafe, ushort, using, virtual, volatile, void i while.

Кључне речи представљају резервисане индентификаторе који имају одговарајуће значење и које користи језик C#. Индентификатори представљају имена именских простора, класа, метода, променљивих, итд.
[2]

2
GMap БИБЛИОТЕКА

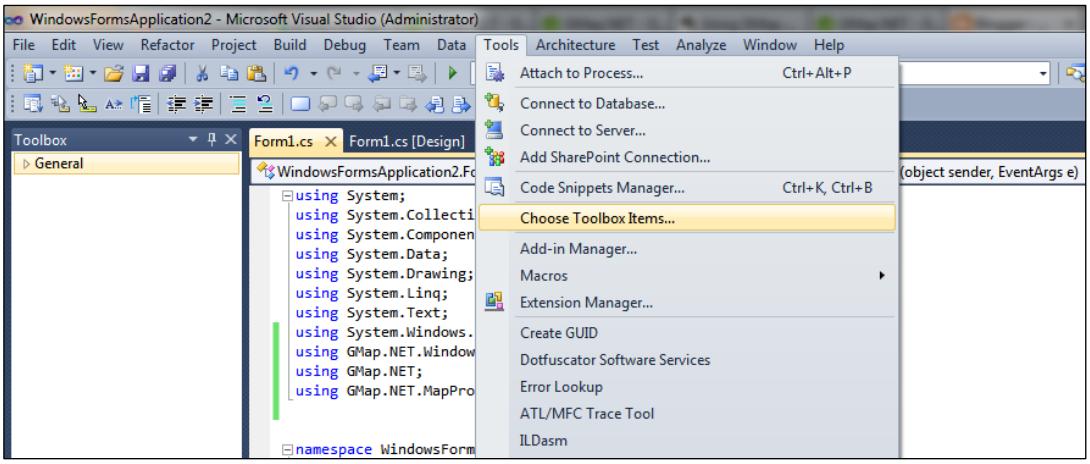
У наредном пројекту је коришћена google мапа, зато што представља цео свет. У задатку се тражи могућност повезивања било која два града у свету и то је разлог одлуке да се у све то уврсти google мапа, са својим библиотекама. Мапом се манипулише зумирањем а концепт повезивања два града линијом није било потребе разрађивати, јер сама мапа у себи то већ садржи.

GMap.NET је бесплатна NET контрола која омогућава функционалност мапирања у апликацији. Може се користити у Winform-и , VPF и мобилним апликацијама и подржава више карата укључујући OpenStreetMap.

- OpenStreetMap
- Yahoo Maps
- Bing Maps
- ArcGIS
- Google

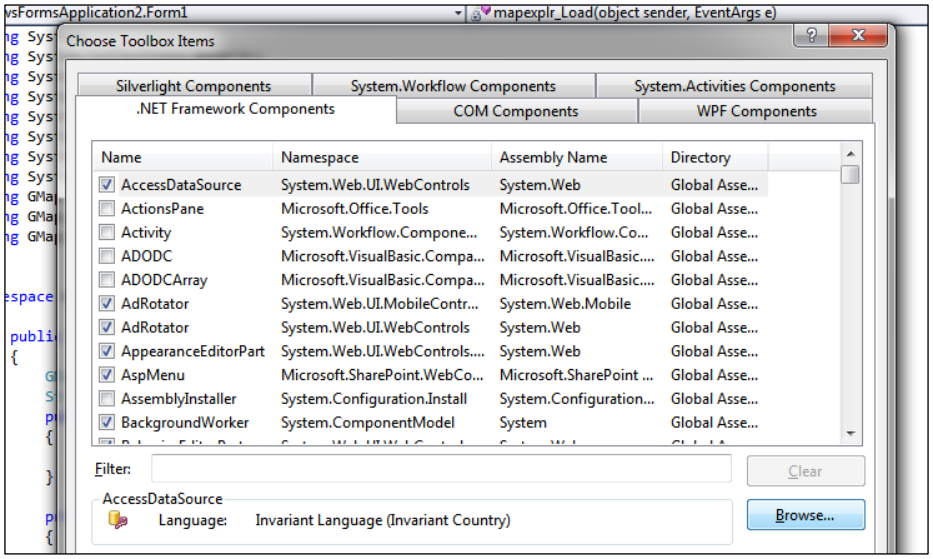
У скинутој датотеци наилази се на датотеку са неколико фајлова, од којих су најбитнији *GMap.NET.Core.dll* і *GMap.NET.WindowsForms.dll.* Следи сликовито објашњење додавања ових библиотека у пројекат.

Прво је неопходно додати *GMap.NET.WindowsForms.dll fajl* у *Visual Studio IDE Toolbox.* На наредној слици под бројем 10 је и сликовито приказан поступак.



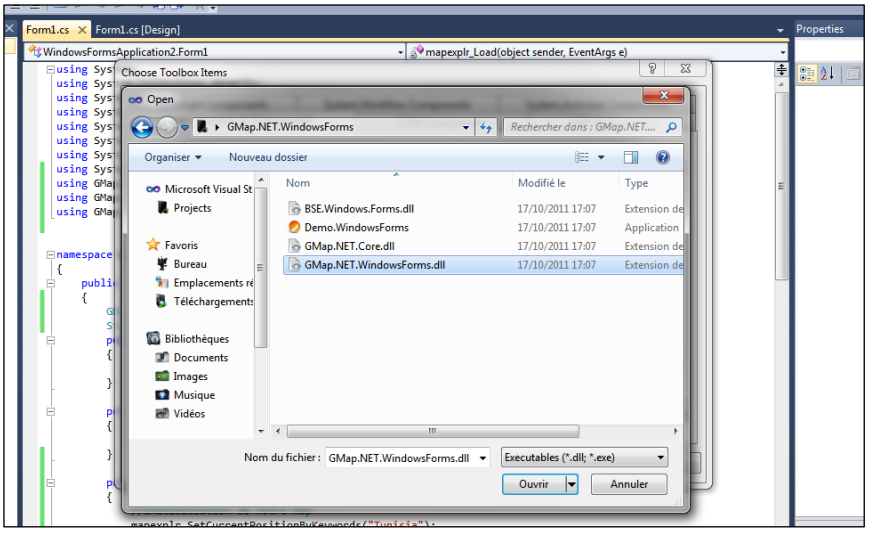
Слика 10.

Додавање GMap.NET.WindowsForms.dll fajl y Visual Studio IDE Toolbox. Након одабира горе наведене опције, требало би да се појави слика 11.



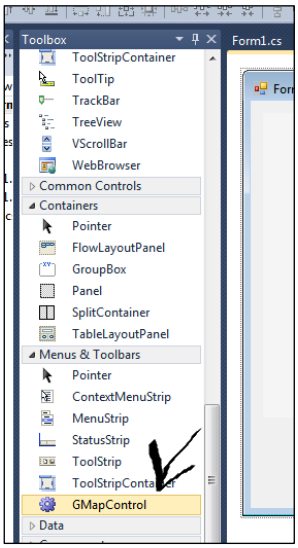
Слика 11.

Пре него што се кликне на дугме *Browse*, треба одабрати *.NET Framework Components* tab. Кликом на дугме *Browse*, отвара се универзални програм за *upload* фајлова, где је потребно одабрати *GMap.NET.WindowsForms.dll* фајл, затим дугме *Open*, а онда *OK*, као што је приказано на слици 12.

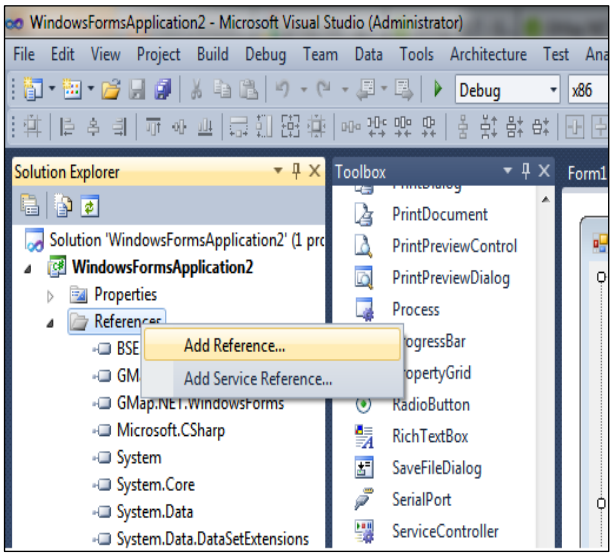


Слика 12.

Након одрађених свих ових корака, на табу *Toolbox* би требала да се појави нова опција- *Gmap control* (Слика 13.) и након тога треба додати референце за нове библиотеке. То се може лако урадити, пратећи упутства са слике 14.



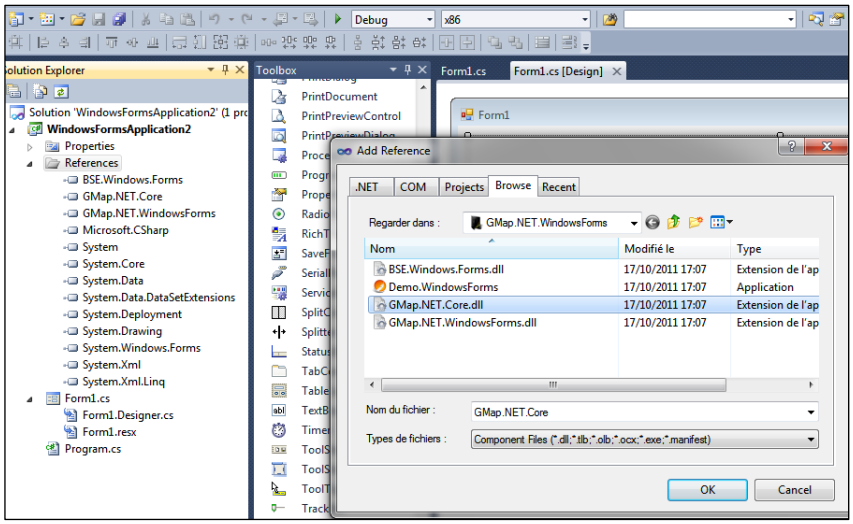
Слика 13.



Слика 14.

Поступак додавања референци је исти као и при додавању *GMap.NET.WindowsForms.dll* фајла. Та референца у ствари представља већ поменути други битан фајл- *GMap.NET.Core.dll*.

Након додавања два већ наведена фајла у VisualStudio, може се без проблема кренути са израдом пројекта. [3]



Слика 15.

3
ПРОМЕНЉИВЕ

Променљива представља именовану локацију у RAM меморији рачунара која чува вредност одређеног типа. Свака променљива мора бити јединствено означена. Именовање променљиве је сасвим произвољно у оквиру комбинације дозвољених знакова. Ипак постоје нека генерална правила

Не саветује се именовање променљивих које се разликују само према великим и малим словима а саветује се да променљива почне малим словом. Постоје два основна правила за именовање променљивих:

1. Мађарска нотација- Hungarian notation
2. Камиља нотација- Camel notation

Camel notation именује промељиве малим словима а велика се користе код имена од више речи да означе почетак следеће речи са великим словом. Примери ове нотације су:

- првиПодатак
- новаГодина
- првиМај
- новиСад
- новиСадСпенс

Hungarian notation почиње са ознаком типа променљиве после чега следи назив променљиве. Саветује се употреба Camel notation а не и Hungarian notation. Имена индентификатора су такође важна за сарадњу са другим .NET језицима која поставља већа ограничења него сам C#. C# је строго типизиран језик. Сваки индентификатор је неопходно декларисати пре употребе. Декларацијом се наводи тип и име променљиве.

Пример декларације:

```
int age;
```

Додељивање вредности декларисаној промељивој:

```
age = 42;
```

Коришћење променљиве:

```
Console.WriteLine(age);
```

Тип податка	Опис	Пример
int	Цели бројеви	int x = 7;
long	Цели бројеви са већим опусом	long x = 42L;
float	Реалан број једноструке тачности	float x = 0,42F
double	Реалан број двоструке тачности	double x = 0,42
decimal	Децимални бројеви	decimal x = 2,5
string	Низ карактера	string x = “summer“;
char	Један карактер	char x = “s”;
bool	Логично/тачно/логично нетачно(true/false)	bool x = false;

3.1. Променљиве за форму

За декларисање променљивих које су употребљене за креирање форме, коришћена је Camel notation. Претежно су int tipa. Форма представља оквир целокупне апликације,а поред свега има и своје димензије. Почетне димензије су на нули, а након покретања „узимају“ се информације о резолуцији екрана, и о величини апликације.

У менију за додељивање компоненти, подешене су ширина и дужина форме, које износе 816x539 piksela. Ове информације се „узимају“ након покретања. Овакав начин додељивања величине, која је на почетку нула, а касније добија димензије је био неопходан због прегледног распореда и онда када корисник одлучи да maximize-ује прозор, тј. форму апликације. Између осталих, постоји и променљива типа

Color, која боји сепаратор и оквире text box-ева, и променљива која дефинише дебљину оквира и сепаратора.

```
1:    //****************FORM VARIABLE************************//
2:
3:    private int screenWidth = 0, screenHeight = 0;
4:    private int appWidth = 0, appHeight = 0;
5:    private int lineX = 50;
6:    private Color separatorColor = Color.DodgerBlue, borderColor =Color.DodgerBlue, txtBorderColor = Color.DodgerBlue;
7:    private int separatorSize = 1, txtBorderSize = 2;
```

3.2. Променљиве за GMap

И ове променљиве су декларисане по Camel notation правилу, и користе се за смештање информација које се касније приказују на форми. Помоћу променљивих типа: string, start i end, уноси се прва и друга локација. Default-ни zoom мапе је 15, и касније се мења, у зависности од релације.

При покретању апликације, мапа ће увек бити фокусирана на Београд, а то се касније мења, у зависности од изабраних локација.

Међу наведеним променљивима су и оне којима се додељују боја и дебљина линија које повезују градове. [4]

```
10:    //********MAPS VARIABLE*********************************//
11:
12:    private String start = "", end = "";
13:    private int zoom = 15;
14:    private String pocetniGrad = "Beograd, Serbia";
15:    private int drivingLine = 5, airLine = 5;
16:    private Color drivingColor = Color.Blue, airColor = Color.Red;
17:    String[] info;
18:    int count = 1;
```

4
METODE

Методе се састоје из потписа и тела методе. Под потписом се подразумева модификатор приступа, затим повратни тип, назив саме методе и улазни параметри. Тело методе садржи саму функционалност методе, односно одређени посао који она одрађује. Поред свега осталог, треба додати то да C# користи типизиране податке, што значи да нека променљива не може да садржи било који тип података, као што је могуће у php-у на пример, већ је свака променљива одређеног типа и сходно томе садржи податке само одређеног типа.

Једна од објектно оријентисаних могућности коју имплементира C#, је такозвано преоптерећење метода (overload), односно ситуација да неколико метода назовемо истим именом, а да имају другачије потписе. Поменуто преоптерећење метода се често користи у ситуацији када треба да се изврши иста операција.

4.1. Методе за форму

IDE даје програмеру прилично јасну слику о догађајима које може да придружи некој контроли или форми. На прозору Properties постоји дугме Events. Када се кликне на то дугме добије се списак догађаја који су придружени контроли или форми која је селектована.

4.1.1 Метода ***OnPaint***

OnPaint метода се извршава у тренутку покретања апликације и служи за исцртавање сепаратора и оквира на форми и текстуалном пољу. Прво се креира оловка myPen1 којој се кроз конструктор класе проследи жељена боја Color.Blue, а затим се подеси дебљина линије. Након тога позива се метода за исцртавање линије DrawLine(pen, startX, startY, stopX, stopY) која као улазне параметре има објекат оловке који је претходно декларисан, има X и Y почетну тачку одакле ће кренути исцртавање линије и X и Y, крајњу тачку где ће се завршити линија.

Код исцртавања оквира процедура око креирања оловке је иста, али уместо исцртавања линије потребно је исцртати квадрат око компоненте код које се ставља оквир. Да би се нацртао квадрат мора се позвати метода DrawRectangle(pen, x, y, width, height) која као улазне параметре има објекат оловке који је претходно декларисан, има X и Y почетну тачку одакле ће кренути исцртавање квадрата, па тако има и дужину и ширину квадрата.

```
1:          protected override void OnPaint(PaintEventArgs e)
2:            {
3:                base.OnPaint(e);
4:                Graphics g;
```

```
    5:
    6:                 g = e.Graphics;
    7:
    8:                 Pen myPen1 = new Pen(separatorColor);
    9:                 myPen1.Width = separatorSize;
   10:                 g.DrawLine(myPen1, 25, lineX, appWidth-40,
lineX);
   11:
   12:                 Pen myPen2 = new Pen(separatorColor);
   13:                 myPen2.Width = separatorSize;
   14:                 g.DrawLine(myPen2, 166, 90, 166, appHeight -
80);
   15:
   16:                 Pen myPen7 = new Pen(txtBorderColor);
   17:                 myPen7.Width = txtBorderSize;
   18:                 g.DrawRectangle(myPen7,
txtFirstPlace.Location.X, txtFirstPlace.Location.Y,
txtFirstPlace.Width, txtFirstPlace.Height);
   19:
   20:                 Pen myPen8 = new Pen(txtBorderColor);
   21:                 myPen8.Width = txtBorderSize;
   22:                 g.DrawRectangle(myPen8,
txtSecondPlace.Location.X, txtSecondPlace.Location.Y,
txtSecondPlace.Width, txtSecondPlace.Height);
   23:             }
```

4.1.2 Метода ***Form1_Load***

Form1_Load метода се извршава онда када се покрене апликација. То је метода која служи за иницијализацију форме и мапе. Пример кода се може видети у наредном тексту.

```
1:          private void Form1_Load(object sender, EventArgs
e)
2:          { //Kada se pokrene forma vrsi se inicijalizacija
forme i mape
3:              init();
4:              mapInit();
5:          }
```

4.1.3 Метода ***Form1_Resize***

Form1_Resize метода има улогу да проверава када корисник промени величину форме, и након тога одради одређена израчунавања. Када се деси промена величине форме ова метода прво узима вредности од нове величине и на основу њих поставља величину мапе и лабела на одговарајућу величину, а такође и подешава почетну локацију лабела који се налазе на доле десној страни из разлога што је потребно да се подесе увек на истој позицији гледајући од краја форме. Ово је битно јер су лабели по default-у постављени на неку локацију а када дође до промене величине и они морају бити померени како би апликација изгледала лепо. Пример кода је приказан у наредном тексту.

```
1:          private void Form1_Resize(object sender,
EventArgs e)
2:          { //Kada se resajzuje forma resajzuju se i
separatori, mapa i tekstualna polja za informacije
3:              appWidth = Width;
4:              appHeight = Height;
5:
6:              gmap.Size = new Size(appWidth - 245,
appHeight - 190);
7:              lblInfo1.Size = new Size(lblInfo1.Width,
appHeight - 400);
8:              lblInfo2.Location = new Point(1, appHeight -
70);
9:
10:             Invalidate();
11:         }
```

4.1.4 Метода ***msg***

Помоћу ове методе може се много брже позвати метода која је задужена за приказ поруке на екрану. Метода **msg** има улазни параметар који је типа **String**, и помоћу њега прослеђујемо поруку коју желимо да се прикаже. Када се позове већ поменута метода, она позива другу методу, тј методу **MessageBox.Show** и њој прослеђује поруку која би требало да буде приказана.

```
1:          private void msg(String msg)
2:          { //Ova metoda omogucava da brze pozivamo
MessageBox
3:              MessageBox.Show(msg);
4:
```

4.1.5 Метода ***init***

Следећа метода је задужена за узимање информација о резолуцији екрана, за узимање информација о величини апликације, и на основу тих информација подешава остале компоненте на форми. Метода садржи и команду која забрањује фокус на компоненте **RadioButton**, то заправо значи да када корисник притиска тастер **TAB** ова компонента никада неће бити означена. Метода **init** се покреће када се покрене и апликација и њена улога је да на основу резолуције екрана и величине апликације одреди колико ће велика бити мапа и где ће се налазити лабели.

Величина мапе се израчунава тако што се од тренутне ширине апликације одузме 245 пиксела и од тренутне висине апликације одузме 190 пиксела. Овим израчунавањем ће мапа увек бити исто удаљена од ивице форме независно од њене величине.

Исти принцип израчунавања се пимењује и за позиционирање label компоненте на форму. Пример методе се може видети у наредном тексту.

```
1:          private void init()
2:          { //Uzimanje informacija o ekranu
3:              rbDrivingLine.TabStop = false; //Zabrana
fokusa na ovu komponentu
4:              rbAirLine.TabStop = false; //Zabrana fokusa
na ovu komponentu
5:
6:              //Uzimanje informacija o rezoluciji ekrana
7:              screenWidth =
Screen.PrimaryScreen.Bounds.Width;
8:              screenHeight =
```

```
Screen.PrimaryScreen.Bounds.Height;
   9:
  10:                 //Uzimanje informacija o velicini aplikacije
  11:                 appWidth = this.Width;
  12:                 appHeight = this.Height;
  13:
  14:                 //Podesavanje velicine mape na ekranu
  15:                 gmap.Size = new System.Drawing.Size(appWidth
- 245, appHeight - 190);
  16:
  17:                 //Podesavanje velicine tekstualnih polja za
informacije
  18:                 lblInfo1.Size = new Size(lblInfo1.Width,
appHeight - 400);
  19:                 lblInfo2.Location = new Point(1, appHeight -
70);
  20:             }
```

4.2 Методе за *TEXT BOX*

Методе које су битне на компоненту **TextBox** имају улогу да проверавају када је фокусирана компонента, које је дугме на тастатури и ако детектује да је корисник притиснуо ENTER позива методе које врше претрагу унетих градова. Поред ове методе користи се још једна метода а она има улогу да провери када корисник кликне мишем на **TextBox**, и када се то деси долази до брисања текста из поменуте компоненте.

4.2.1 Метода *txtFirstPlace_KeyDown i txtSecondPlace_KeyDown*

Ове две методе представљају листенере за компоненте **TextBox**. Обе методе имају исти код који се извршава онда када корисник притисне неко дугме на тастатури. Када се то деси **if** петљом се проверава да ли је кликнуто дугме ENTER и уколико јесте, позива се метода **work** која служи за повезивање градова.

```
   1:             private void txtFirstPlace_KeyDown(object sender,
KeyEventArgs e)
   2:             { //Ova metoda detektuje pritisak nekog tastera
na tastaturi dok je fokusiran txtFirstPlace
   3:                 if (e.KeyCode == Keys.Enter)
```

```
4:                { //Provera se da li je pritisnuti taster
ENTER
5:                    e.SuppressKeyPress = true;
6:                    work();
7:                }
8:            }
```

4.2.2 Метода *txtFirstPlace_MouseClick i txtSecondPlace_MouseClick*

Ове методе су такође листенери који се активирају када корисник кликне левим кликом миша на компоненту **TextBox**. Када се то деси позива се команда која брише постојећи текст са кликнуте компоненте.

```
1:            private void txtFirstPlace_MouseClick(object
sender, MouseEventArgs e)
2:            { //Klikom na TextBox za prvu lokaciju brise se
njegov sadrzaj
3:                txtFirstPlace.Text = "";
4:            }
```

4.3 Методе за *RADIO BUTTON*

Методе за **RadioButton** имају улогу да позивају методе или за ваздушно или копнено повезивање градовам у зависности који је **RadioButton** штиклиран. Поменута компонента је добра када кориснику дајемо могућност да изабере само једну од више понуђених опција, зато што је небитно колико ових компоненти имамо на форми када се једна укључи , све остале се искључе, што је веома погодно и игра важну улогу са одабиром ваздушног или копненог повезивања.

4.3.1 Метода *rbDrivingLine_CheckedChanged i rbAirLine_CheckedChanged*

Ова метода се активира само онда када се промени стање rbDrivingLine **RadioButtona**. Тачније када се штиклира ова компонента позива се код, који врши даљу проверу и који повезује изабране градове.

```
1:            private void rbDrivingLine_CheckedChanged(object
sender, EventArgs e)
2:            { //Nakon stikliranja ovog CheckBoxa poziva se
metoda work
3:                rbDrivingLine.TabStop = false;
4:                work();
5:            }
```

4.4 Метода за *TIMER*

Метода за **Timer** има могућност да се активира и да у одређеном временском интервалу позива одређене методе све док се не деактивира.

4.4.1 Метода ***tShowInfo_Tick***

Метода **tShowInfo_Tick** је подешена тако да се након њеног активирања позива на сваких пет секунди, и при сваком позиву има улогу да прикаже кориснику информације о улицама кроз које све треба проћи од првог до другог града. Ова метода приказује по једну улицу на сваких пет секунди и поставља текст у **Label**-у на форми. Метода повећава бројач и вади из листе са улицама, улицупод редним бројем од вредности бројача, приказује је на форми, а уколико стигне на крај листе, бројач враћа на нулу и стопира **Timer**. Пример кода је приказан у наредном делу рада.

```
   1:          private void tShowInfo_Tick(object sender,
EventArgs e)
   2:          { //Ovaj tajmer na svakih pet sekundi pokazeju po
jednu ulicu kroz koju se mora proci od pocetka do kraja
destinacije
   3:              lblInfo2.Text = info[count];
   4:
   5:              if(count == info.Length)
   6:              {
   7:                  tShowInfo.Stop();
   8:                  count = 0;
   9:              }
  10:
  11:              count++;
  12:          }
```

4.5. Методе за *GMap*

Методе за Gmap апликацију су од кључног значаја зато што оне управљају са Gmap библиотеком и са разним методама које нам омогућавају повезивање две локације у целом свету. Све те поменуте методе ће даље бити разрађене у даљем делу текста.

4.5.1 Метода ***mapInit***

Ова метода се позива приликом покретања апликације и она иницијализује мапу. Такође поставља, да је default-на мапа google map-а и брише сва остала поља. Следи пример кода.

```
   1:          private void mapInit()
   2:          { //Inicijalizacija google mape
   3:              gmap.MapProvider =
GMap.NET.MapProviders.GoogleMapProvider.Instance;
   4:              GMap.NET.GMaps.Instance.Mode =
GMap.NET.AccessMode.ServerOnly;
   5:
   6:              clearAll();
   7:          }
```

4.5.2 Метода ***search***

Ова метода пре свега брише све путање са мапе а затим узима локације од првог и другог изабраног града. У овој методи постоји једна **if** петља којом се проверава који је **RadioButton** означен и која у својим блоковима има **try catch** блок који позива цртање линије на мапи а уколико дође до неке грешке, овај блок не дозвољава програму да прекине са радом већ наставља даље.

```
   1:          private void search(String prviGrad, String
drugiGrad)
   2:          { //Ova metoda uzima zeljene lokacije, proverava
koji je CheckBox stiukliran i na osnovu togra poziva odredjene
metode koje crtaju liniju i prikazuju informacije i zumiraju
mapu
   3:              gmap.Overlays.Clear();
   5:              PointLatLng? pos1 = getPosition(prviGrad);
   6:              PointLatLng? pos2 = getPosition(drugiGrad);
   7:
   8:              if (rbDrivingLine.Checked == true)
   9:              { //Ukoliko je stikliran autic radice se blok
ispod
  10:                  try
  11:                  {
```

```
12:                         setDrivingLine(prviGrad, drugiGrad,
zoom);
13:                     }
14:                     catch (Exception e3)
15:                     {
17:                     }
18:                 }
19:                 else if (rbAirLine.Checked == true)
20:                 { //Ukoliko je stikliran avioncic radice se
blok ispod
21:                     try
22:                     {
23:                         setAirLine(pos1.Value.Lat,
pos1.Value.Lng, pos2.Value.Lat, pos2.Value.Lng);
24:                     }
25:                     catch (Exception e4)
26:                     {
28:                     }
29:                 }
30:             }
```

4.5.3 Метода ***setDrivingLine***

Ова метода служи за исцртавање линије по путевима од првог до другог града. Она као улазне параметре има име првог града, име другог града и zoom.

Ова метода функционише тако што се прво креира објекат **route** и у њега се кроз конструктор класе унесу први град, други град и zoom. Након тога потребно је направити објекат за линију која повезује градове и доделити му боју и дебљину. А након свега тога долази до постављања линије и зумирања мапе. Ове команде се налазе у try catch блоку и уколико дође до било какве грешке програм ће наставити са радом али ће приказати поруку на екрану `"GRESKA: Pokusajte ponovo!"`.

У наредном тексту приказан је код претходно објашњене методе.

```
1:                 private void setDrivingLine(String prviGrad,
String drugiGrad, int zoom)
2:             { //Ova metoda iscrtava liniju po putevima od
prvog do durugog grada
3:                 MapRoute route =
```

```
GMapProviders.GoogleMap.GetRoute(prviGrad, drugiGrad, false,
false, zoom);
    4:
    5:                try
    6:                {
    7:                    GMapRoute r = new GMapRoute(route.Points,
"My route");
    8:                    r.Stroke.Width = drivingLine;
    9:                    r.Stroke.Color = drivingColor;
   10:
   11:                    GMapOverlay routesOverlay = new
GMapOverlay("routes");
   12:                    routesOverlay.Routes.Add(r);
   13:                    gmap.Overlays.Clear();
   14:                    gmap.Overlays.Add(routesOverlay);
   15:                    gmap.ZoomAndCenterRoute(route);
   16:                }
   17:                catch (Exception e1)
   18:                { //Ukoliko ne postoji lokacija koja je uneta
ili ne postoji ni jedan put kojim bi mogli da se povezu gradovi
izbacice gresku ispod
   19:                    msg("GRESKA: Pokusajte ponovo!");
   20:                }
   21:            }
```

4.5.4 Метода ***setAirLine***

Ова метода исцртава линију ваздушним путем од првог до другог града. Она креира полигон али само између две тачке на мапи и обоји га у црвено. Ова метода има четири улазна параметра од којих два представљају локацију првог града а друга два локацију другог града на мапи. Ове две информације везане за један град се зову **latitude** и **longitude** и оне чине једну тачку на мапи, и представљају координате на google map-и. Код за ову методу је приказан у наредном тексту.

```
    1:          private void setAirLine(double lat1, double lng1,
double lat2, double lng2)
```

```
2:          { //Ova metoda iscrtava liniju vazdusnim putem od prvog do drugog grada
3:              GMapOverlay polyOverlay = new GMapOverlay("polygons");
4:
5:              List<PointLatLng> points = new List<PointLatLng>();
6:              points.Add(new PointLatLng(lat1, lng1));
7:              points.Add(new PointLatLng(lat2, lng2));
8:
9:              GMapPolygon polygon = new GMapPolygon(points, "mypolygon");
10:             polygon.Fill = new SolidBrush(Color.FromArgb(50, Color.Red));
11:             polygon.Stroke = new Pen(airColor, airLine);
12:
13:             polyOverlay.Polygons.Add(polygon);
14:             gmap.Overlays.Add(polyOverlay);
15:             gmap.ZoomAndCenterRoute(polygon);
16:         }
```

4.5.5 Метода ***getPosition***

Ова метода има један улазни параметар који представља име локације и има излазни параметар који представља географску локацију на мапи.

```
1:          private PointLatLng? getPosition(String place)
2:          { //Ova metoda na osnovu unetog imena grada vraca geografsku lokaciju (Latitude/Longitude)
3:              GeoCoderStatusCode status = GeoCoderStatusCode.Unknow;
4:              PointLatLng? pos = GMapProviders.GoogleMap.GetPoint(place, out status);
5:
6:              return pos;
7:          }
```

4.5.6 Метода ***getInfo***

Ова метода има улогу да враћа све информације о изабраним градовима. Она проверава да ли је чекирана копнена за копнену линију, или ваздушна за ваздушну линију, и на основу тога показује различите информације.

```
 1:          private String[] getInfo(String prviGrad, String
drugiGrad, double lat1, double lng1, double lat2, double lng2)
 2:          { //Ova metoda vraca sve informacije sezane za
izabrane gradove
 3:              String[] str = new String[1000];
 4:              GDirections s;
 5:
 6:              var x =
GMapProviders.GoogleMap.GetDirections(out s, prviGrad,
drugiGrad, false, false, true, false, false);
 7:              if (x == DirectionsStatusCode.OK)
 8:              {
 9:
10:                  String km = miToKm(s.Distance);
11:
12:                  lblInfo1.Text += s.StartAddress + "\r\n"
+
13:                                   s.EndAddress + "\r\n";
14:
15:                  if (rbDrivingLine.Checked == true)
16:                  { //Ukoliko je stikliran autic radice se
blok ispod
17:                      lblInfo1.Text += s.Summary + "\r\n" +
18:                                       s.Distance +
"le\r\n" +
19:                                       km.ToString() + "
km\r\n" +
20:                                       s.Duration + "\r\n";
```

```
 21:
 22:                         int br = 0;
 23:                         foreach (var step in s.Steps)
 24:                         { //Izlistava sve ulice kroz koje se
mora proci do zeljene lokacije
 25:                             String streatRouteHTML =
step.HtmlInstructions;
 26:
 27:                             String pattern = @"<[^>]*>";
 28:                             Regex rgx = new Regex(pattern);
 29:                             String streatRoute =
rgx.Replace(streatRouteHTML, "");
 30:
 31:                             str[br] += streatRoute + "\r\n";
 32:                             br++;
 33:                         }
 34:                     }
 35:                     else if (rbAirLine.Checked == true)
 36:                     { //Ukoliko je stikliran avioncic radice
se blok ispod
 37:                         String dist = getDistance(lat1, lng1,
lat2, lng2);
 38:                         lblInfo1.Text += dist + " km\r\n";
 39:                     }
 40:                 }
 41:
 42:                 return str;
 43:             }
```

4.5.7 Метода ***miToKm***

Ова метода има за улогу да претвара миље у километре. Она има улазни параметар који представља миљу, а враћа информацију типа String која представља километре. Ово конвертовање миље у километре се добија тако што се унета миља подели са 0.62137 и тако се добија колико је то километара.

```
    1:          private String miToKm(String mi)
    2:          { //Metoda koja pretvara milju u kilometar
    3:              String km = "";
    5:              try
    6:              {
    7:                  String distance = mi.Replace(" mi", "");
    8:                  String dist = distance.Replace(".", ",");
    9:                  double miles = double.Parse(dist);
   10:                  double kilometar = miles / 0.62137;
   11:                  km = String.Format("{0:0.00}",
kilometar);
   12:              }
   13:              catch (Exception e2)
   14:              {}
   18:              return km;
   19:          }
```

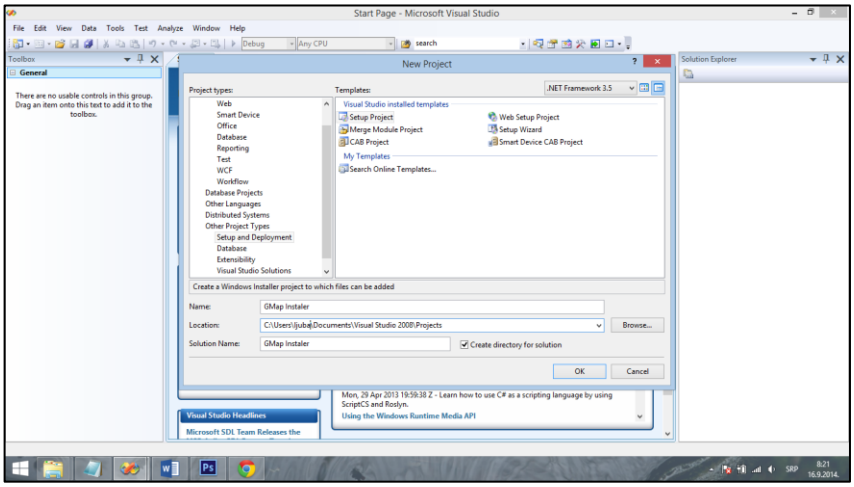
4.5.8 Метода ***work***

Метода **work** је битна метода која узима укуцан први и други град и позива методу која претражује укуцане градове и након тога исцртава линију и позива timer који на сваких пет секунди приказује по једну улицу кроз коју је потребно проћи. [5]

```
    1:          private void work()
    2:          { //Pozivanje metoda koje su bitne za prikaz
linija i informacija
    3:              lblInfo1.Text = "";
    4:
    5:              start = txtFirstPlace.Text;
    6:              end = txtSecondPlace.Text;
    7:
    8:              search(start, end);
    9:
   10:              PointLatLng? pos1 = getPosition(start);
   11:              PointLatLng? pos2 = getPosition(end);
   12:
   13:              try
   14:              {
   15:                  info = getInfo(start, end,
pos1.Value.Lat, pos1.Value.Lng, pos2.Value.Lat, pos2.Value.Lng);
   16:                  lblInfo2.Text = info[0];
   17:                  count = 1;
   18:                  tShowInfo.Stop();
   19:                  tShowInfo.Start();
   20:              }
   21:              catch (Exception e5)
   22:              {
   23:
   24:              }
   25:          }
```

5
Израда инсталације за GMap апликацију

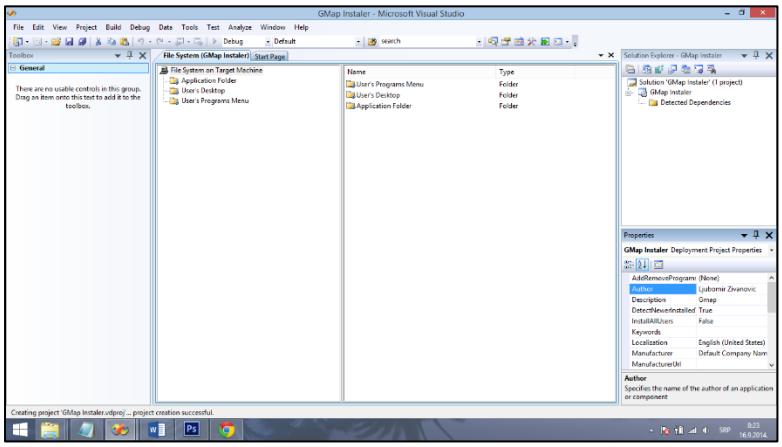
Један од услова задатка био је и посебна инсталација за апликацију. Наредних пар слика приказују поступак креирања пројекта за инсталацију.



Слика 16. Креирање инсталације

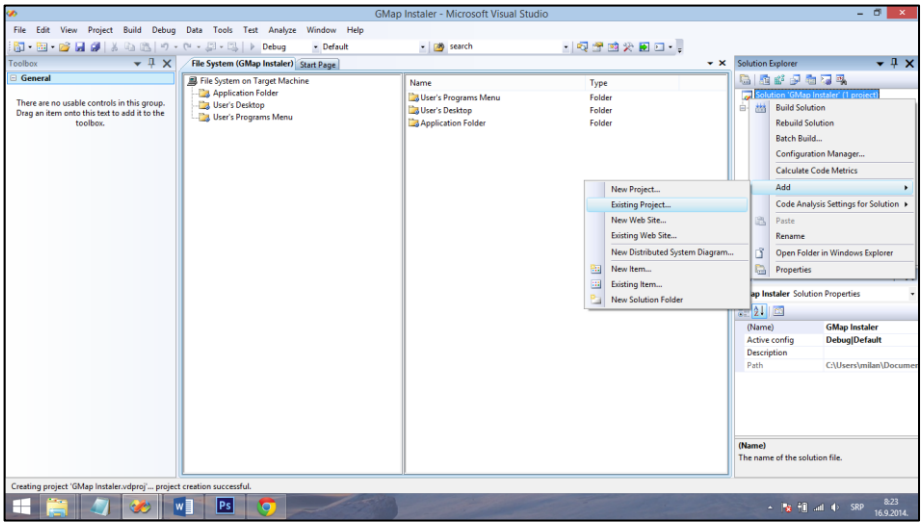
На слици 16 могу се видети почетни кораци приликом креирања инсталације. У menu bar-у Visual Studia може се запазити опција file. Кликом на њу, па на опцију, из падајућег менија, new, а потом на опцију project, отвара се прозор изгледа као на слици 16.

Може се приметити да је из листе опција са леве стране кликнута тема Other Project Types а потом опција- Setup and Deployment. То отвара menu са десне стране, где треба одабрати опцију setup project. Постоји опција, на дну прозора, за подешавање назива пројекта, као и бирање дестинације инсталације, која може остати default-на, а може бити подешена и по слободном избору.



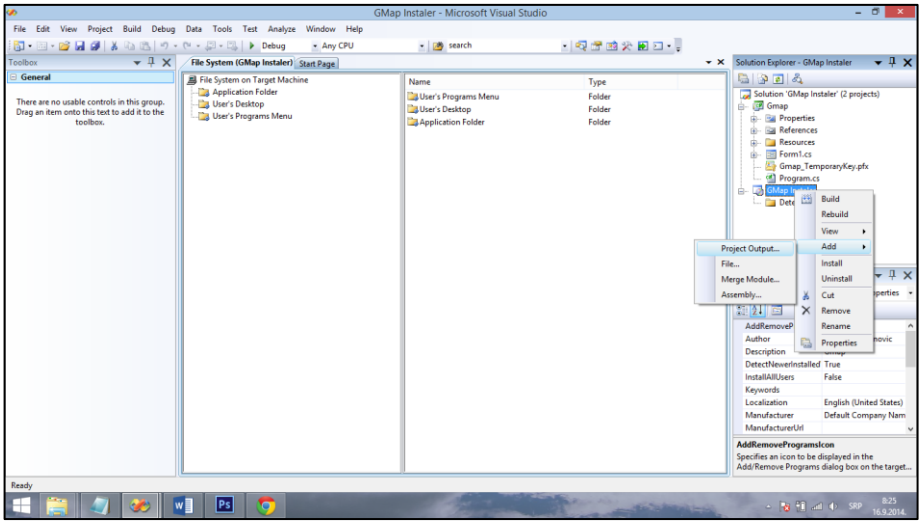
Слика 17. Подешавање

На слици 17. су приказана нека подешавања, као што су име аутора, дескрипција, верзија, програма, итд, којима се приступа преко Solution Explorer-а. Када је све то подешено, следи додавање Gmap пројекта, кликом на прву иконицу у Solution Explorer-у, потом одабиром опције Add, па Exising Project, што је и приказано на слици 18.

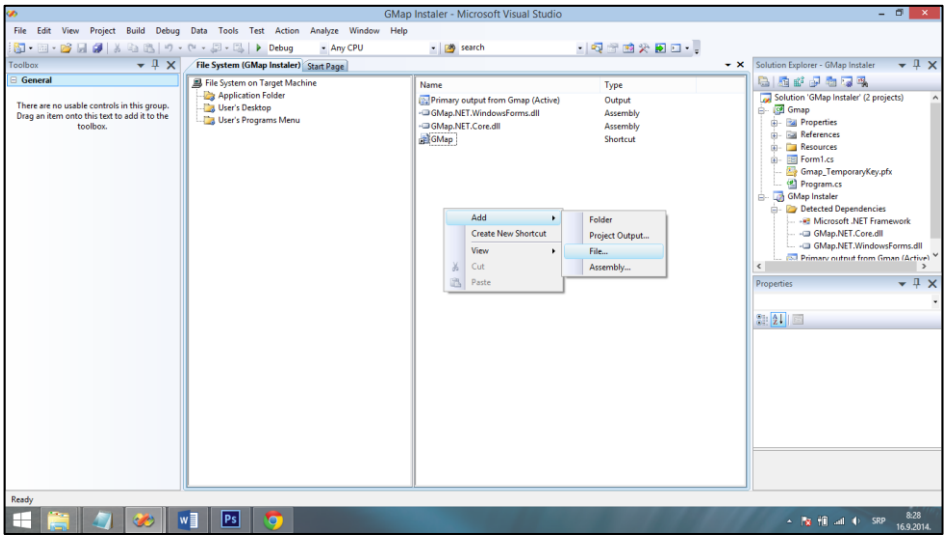


Slika 18. Додавање Gmap пројекта

На слици испод приказано је како се додаје пројекат Gmap у инсталациони пројекат. Тачније додају се фајлови у фолдер који предтавља фолдер Program Files. Када се касније буде инсталирала апликација ови фајлови ће се аутоматски додати у Program Files. Након овога отвориће вам се нов прозор на коме је потребно изабрати опцију *project output* и затим кликнути дугме *OK*.

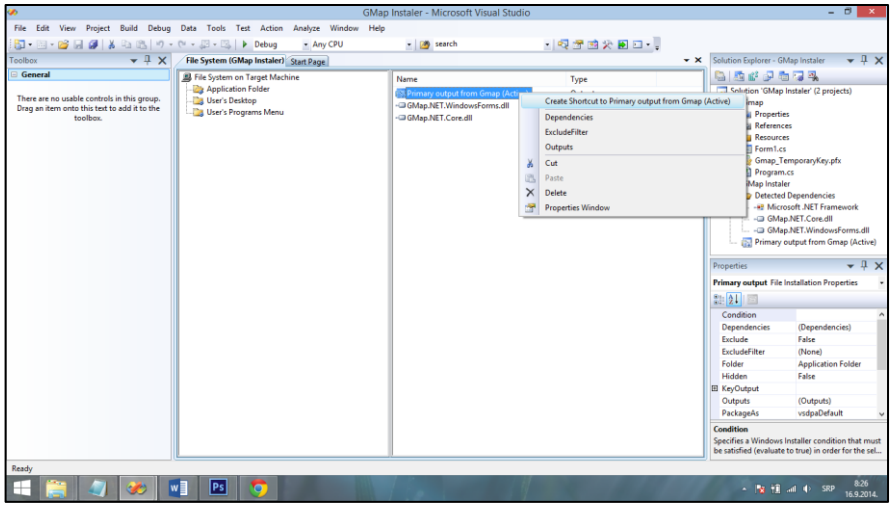


Слика 19. Додавање project output-а



Слика 20. Додавање иконице у пројекат

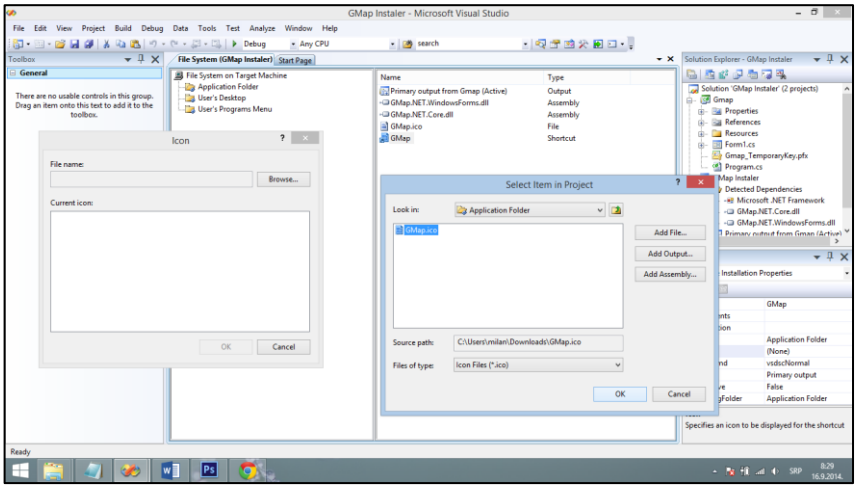
На слици изнад приказано је како се додаје иконица у пројекат за инсталацију. Као што се види потребно је са леве стране кликнути на фолдер *Aplication Folder* а затим на десној страни кликнути десним кликом и изабрати опција *Add* па затим опцију *File*. Након тога отвориће вам се прозор у коме је потребно пронаћи иконицу на хард диску.



Слика 21. Креирање shortcut-а

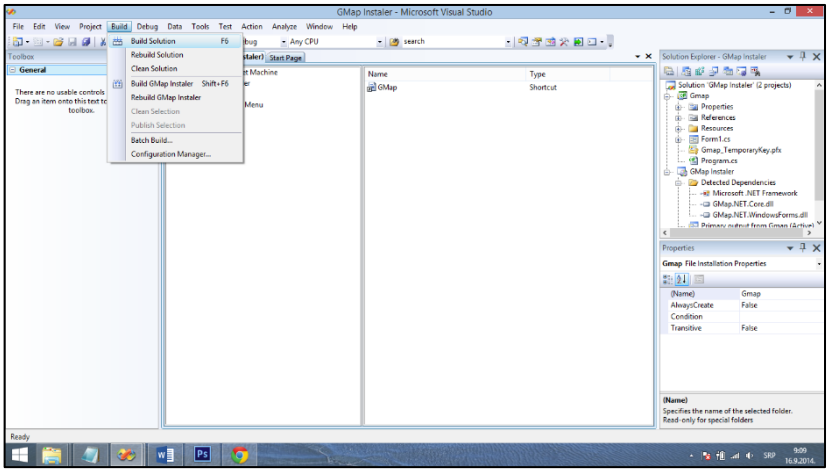
На слици изнад приказано је како се креира *shortcut* апликације. У овом случају потребно је одрадити 2 *shortcut*-а зато што један треба да се прикаже на desktop-у, а други у start menu-у након инсталације.

Када се дода *shortcut* потребно је кликнути на њега а затим у *properties* кликнути на поље *icon*. Ту се отвара нов прозор на коме је потребно кликнути дугме *Browse* а затим се отвара још један прозор где се одабере *Aplication Folder* где се стиклира иконица која је претходно додата у пројекат и након тога кликне дугме *OK*. Овај поступак је потребно одрадити и са другим *shortcut*-ом.



Слика 22. Додавање иконице на shortcut

Када се дође до самог краја креирања инсталације, у последњим корацима је потребно да се један *shortcut* превуче у фолдер *User's Desktop* са леве стране, а други *shortcut* је потребно превући у фолдер *User's Programs Menu,* па у ново креирани фолдер Gmap. То заправо представља иконице са сличицом које ће након инсталације програма бити постављене на desktop-у, и у start menu-jy. Још један последњи корак који је потребан да би се успешно креирао пројекат је тај да се у менију одабере опција *Build* па *Build Solution* што заправо креира инсталациони фајл. [6]



Слика 23. Build Solution

6
ТЕСТИРАЊЕ И ВАЛИДАЦИЈА АПЛИКАЦИЈЕ

Сваки програмски производ па и Gmap треба да прође и кроз фазу тестирања и валидације, где апликација треба да покаже да ли ради оно за шта је намењена и да утврди дефекте у апликацији пре него што се стави у употребу. Када се тестира програмирани производ, тада се извршава тест, где се тестирање врши вештачким подацима. Тестирање је генералнији део свих процеса валидације, који укључују и статичке технике валидације. Резултати треба да покажу све могуће грешке и аномалије, и друге поједине информације о апликацији, које би евентуално указале на непредвиђено понашање.

Апликација се покреће коришћењем тест података где се помно прати њено понашање и анализирају сви детаљи који могу да покажу одређено неприкладно понашање, и на крају теста систем треба да покаже да може да обезбеди одређене функционалности, перформансе и да је стабилан у току нормалне употребе.

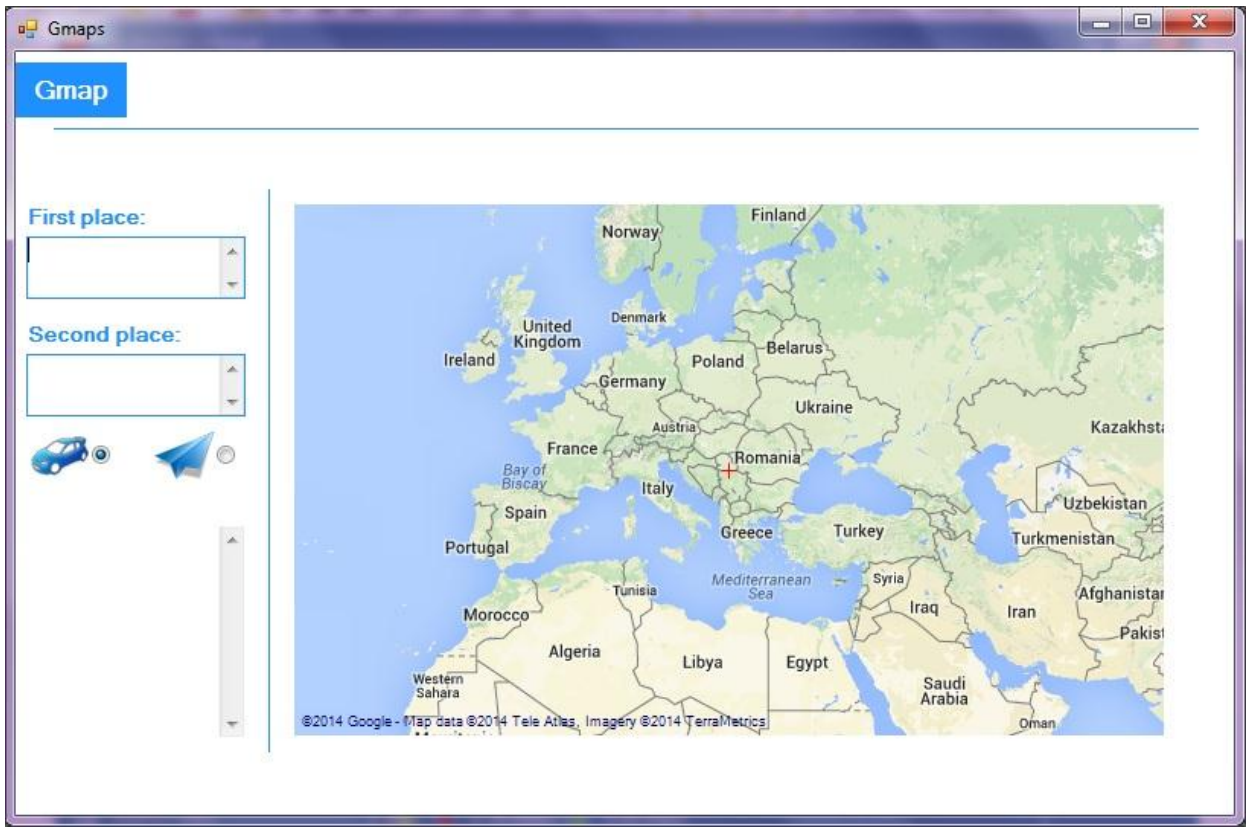
Тестирање апликације у овом случају би требало да открије грешке или отказе, и прикаже да ли постоји неко неприкладно понашање које није дефинисано спецификацијом.Постоје неколико врста тестирања:

- Развојно тестирање-тестира се у току развоја производа да би се открили багови или други дефекти
- Тестирање производа-одређен тим тестира комплетну верзију апликације пре него што се она испоручи у одређеном окружењу за тестирање.
- Корисничко тестирање-корисници, или потенцијални корисници апликације тестирају производ, и овај начин је најбитнији, разлог је тај што корисниково радно окружење има велике ефекте на поузданост, перформансе, употребљивост, итд., а ово се не може рефлектовати у окружењу за тестирање.

На самом почетку тестирања прво се приступа инсталирању апликације, на следећој слици (Слика 24.) се може приметити да се у току инсталације отвара прозор где би требало да се дефинише место смештања података и саме апликације.

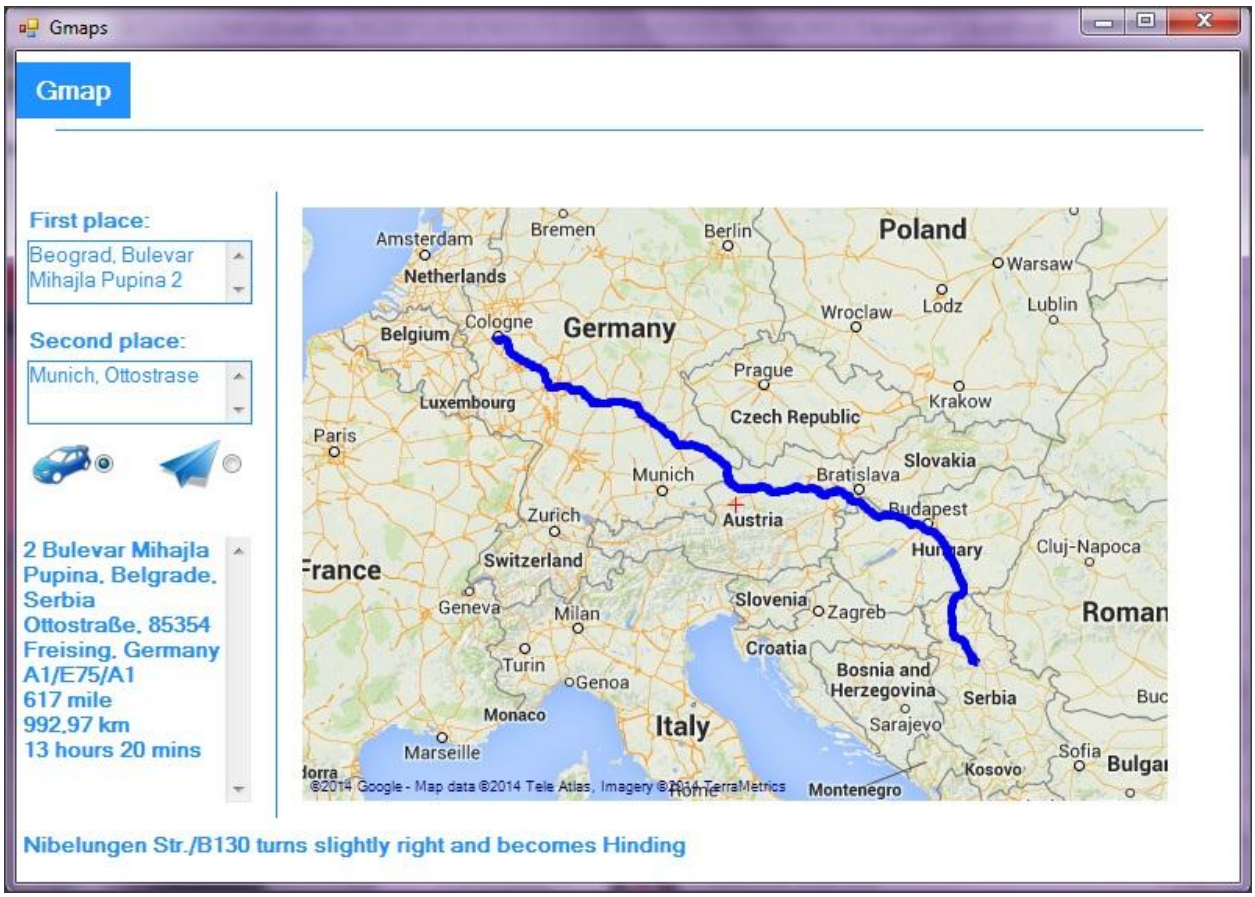
Слика 24. Дефинисање директоријума за смештај фајлова

Након инсталације, појавиће се икона на радној површини рачунара, тј пречица за брзо коришћење названа Gmap. Дуплим кликом на иконицу отвара се прозор апликације као што је приказано на слици 25. Прилично добар део прозора заузима мапа, са фокусом и центром на земљи Србији тј конкретно на граду Београду. Са леве стране налазе се две празнине, тј текстуална поља намењена за упис жељених градова и места. Испод текстуалних поља су смештене две занимљиве иконице за одабир начина повезивања два уписана места.



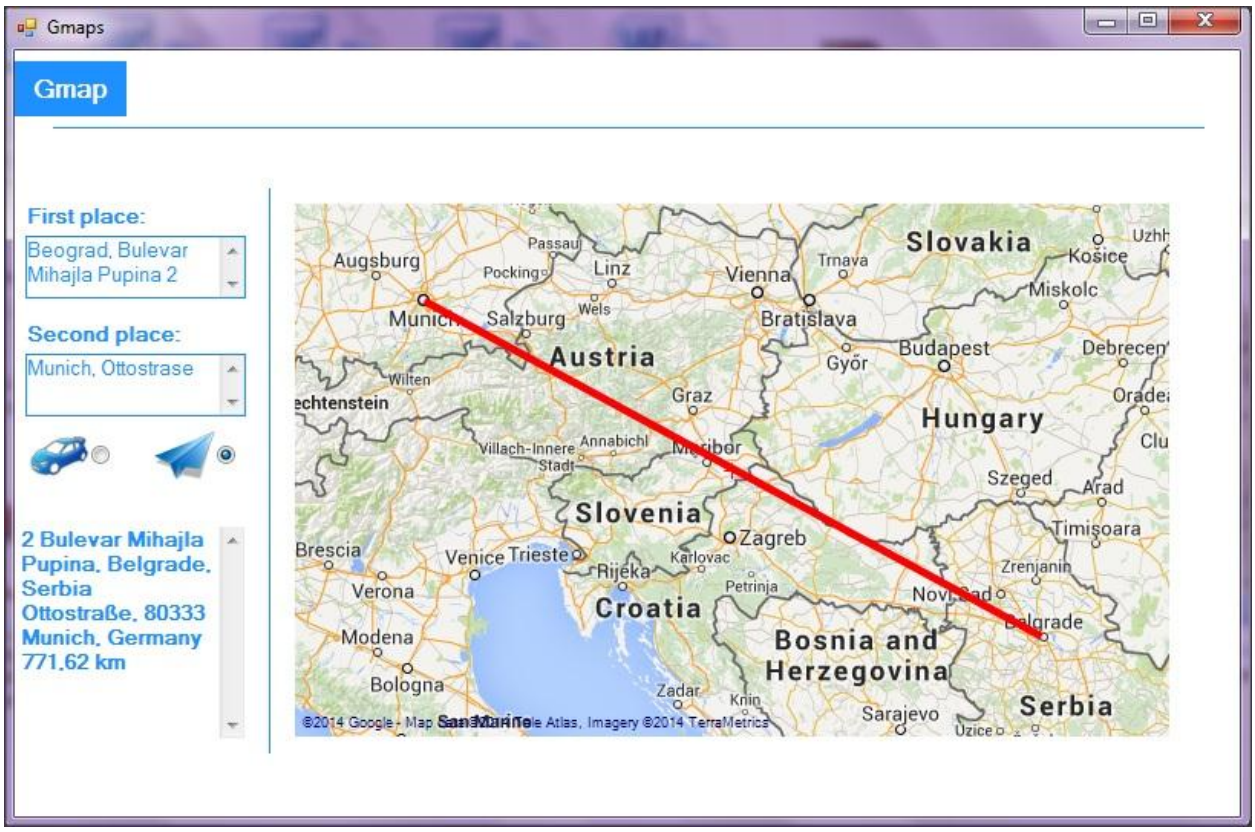
Слика 25. Почетни изглед апликације

Наредна слика приказује један класичан приступ апликацији и тестирању на поједине уобичајене градове и улице. За пример је узет град Београд и град Минхен. Информације исписане са леве стране дају нам увид у то колика је раздаљина између ова два места, и колико је време путовања. Те информације које се виде са леве стране су приказане у случају да је од приказане две иконице изабран аутомобил, тј копнено путовање. Може се приметити и текст који се у дну прозора мења на сваких пет секунди, то је већ поменути Timer, који на сваких пет секунди приказује улице и места кроз које треба да се прође како би се дошло са првог одредишта на коначан циљ, тј друго место.



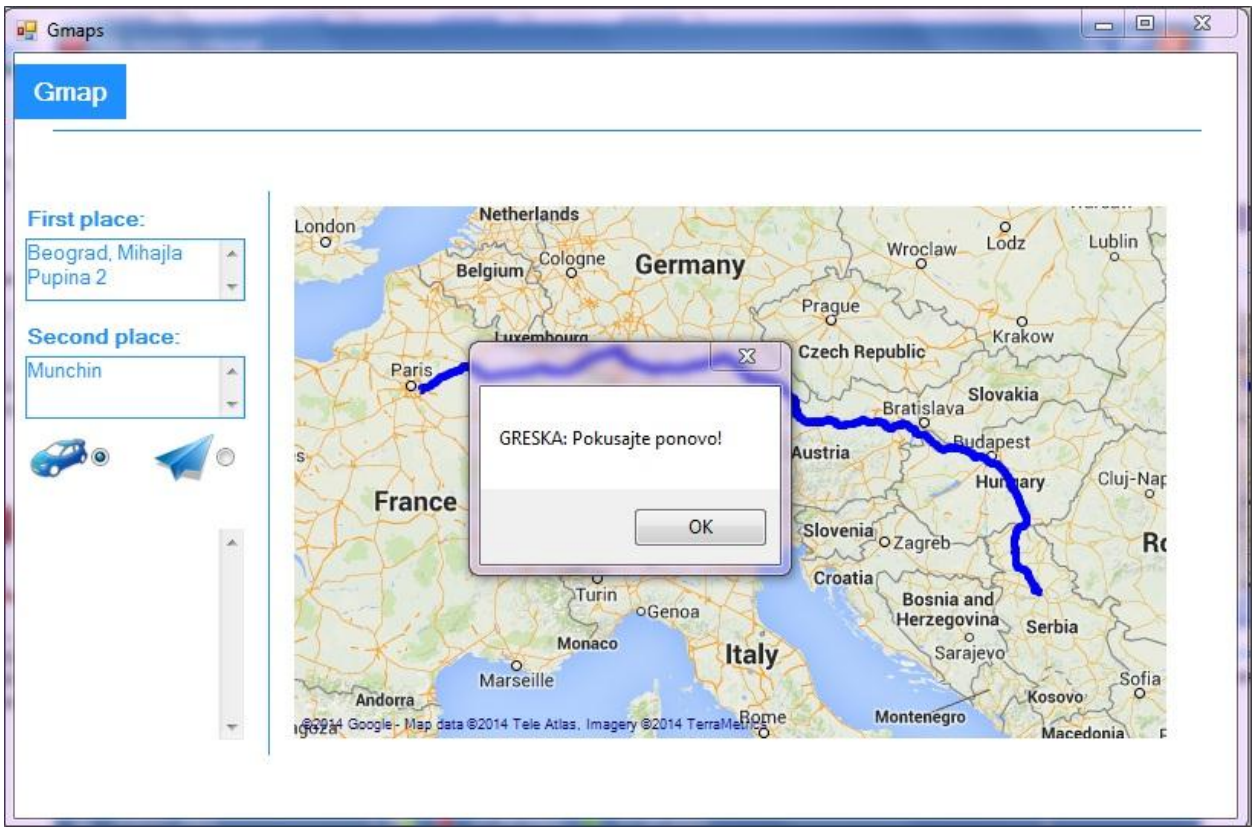
Слика 26. Повезивање градова копненом линијом

У случају да је штиклирана опција за ваздушно путовање, тј авионом, на истој дестинацији као и претходни пример, са леве стране су приказане информације о првој и другој дестинацији, и њиховој раздаљини, а линија која спаја та два места је за разлику од првог примера, црвена права ваздушна линија која их повезује, као што је приказано на слици 27.



Слика 27. Повезивање градова ваздушном линијом

До грешке у апликацији може доћи ако апликација не поседује интернет конекцију, из разлога што користи већ наведене Google мапе, а други проблем је тај да ће програм приказати грешку ако се не упишу прави називи градова, тј ако програм не успе да препозна места о којима се ради. Један такав пример је приказан на слици 28.



Слика 28. Пример погрешно уписаних имена градова

Код валидације се очекује да се апликација понаша исправно коришћењем скупа тестова који опонашају реалну употребу. Валидационо тестирање треба да покаже програмерима и корисницима да апликација обавља све спецификацијама дефинисане задатке, а успешан тест би требало да показује да све функционише онако како би требало. Валидација на крају крајева би требало да представља један процес евалуације апликације или неке компоненте за време или на крају процеса развоја да би се коначно могло утврдити да ли задовољава све захтеве.

Као основни циљ валидације, наводи се успостављање уверења да апликација у потпуности испуњава своју намену.

ЗАКЉУЧАК

Програмирање је и компликовано и једноставно-одвија се у глави и зависи највише од тога да ли знамо шта желимо добити на крају. Уз познавање програмских језика, неопходна је и логика, да би то што желимо спровели у дело, тј. да би могли то да представимо преко кода.

C# програмски језик је најбоље решење када је потребан брз развој desktop апликација за Windows оперативни систем, из разлога што га сматрају једном одличном комбинацијом добрих особина C++ и Jave. Брзина имплементације апликација, добра техничка документација као и распрострањеност Windows оперативног система, чине C# можда и најпопуларнијим програмским језиком данашњице.

Комбиновањем постојећих google map библиотека, и логиком, добијен је код који савршено функционише, и пружа услуге крајњем кориснику. Информације које корисник може видети о градовима, се исписују у реалном времену. Спајање се може извршити копненим и ваздушним путем, у случају да се спајају градови са различитих континената. Може се рећи да развијена апликација удовољава захтевима задатка и испуњава све критеријуме који су непосредно пре израде били задати. Једина неопходност код апликације јесте зависност од интернет конекције, што у данашњем свету не би требало да представља проблем.

ЛИТЕРАТУРА

[1] http://sr.wikipedia.org

[2] C# kroz praktične primere-Charles Wright (Osborne)

[3] http://myactivities-mazen.blogspot.com/2012/01/simple-example-of-gmapnet-with-c.html

[4] http://geekswithblogs.net/saifkhan/archive/2011/08/03/using-gmap.net-ndash-great-maps-for-windows-forms-amp-presentation.aspx

[5] http://www.websofia.com/2013/02/gmap-net-tutorial-maps-markers-and-polygons/

[6] https://software.intel.com/en-us/articles/how-to-creating-your-msi-installer-using-visual-studio-2008