

TravelersAround

Find fellow travelers around you based on your current location - on the go

Nimrod Dayan

Thesis report

Business Information Technology

2011



Business Information Technology

<p>Author or authors Nimrod Dayan</p>	<p>Group or year of entry BIT</p>
<p>Title TravelersAround Find fellow travelers around you based on your current location - on the go</p>	<p>Number of pages and appendices 24 + 75</p>
<p>Supervisor Juhani Välimäki</p>	
<p>The objective of this thesis was to develop an online service for travelers who'd like to meet other travelers while on the go. The development process covered the software analysis, design and implementation of the system.</p> <p>TravelersAround is an online service provided to travelers around the world, who would like to spontaneously meet fellow travelers or even the locals, without any need to plan anything in advance. Many times, travelers decide to go on a trip alone, because none of their friends were available at that time or they couldn't find common interest for the same destination. That's where TravelersAround comes in. As a solo traveler, you would benefit the most out of it. Travelers log onto the system and receive a report of fellow travelers around their current location. The report is shown in real-time and provides up-to-date information.</p> <p>The system was developed in a short time and therefore includes the core features and functions, such as: searching for fellow online travelers in the area, sending and receiving messages, profile status updates and managing friends list. These features will provide the building block of the system's further development. Future development will feature mobile phone application for the popular smart phones in the market.</p> <p>The system was developed in C# programming language, .NET Framework 4.0 and SQL Server 2008 and released under the MIT OSI license. In addition, several third party open source libraries were used (refer to the Appendix 2 : Software Design documentation for the full list).</p>	
<p>Keywords Travelers, Social Network, Location-based, real-time</p>	

Table of contents

Vocabulary	1
Abbreviations.....	1
1 Introduction.....	2
1.1 Background.....	2
1.2 Deliverables.....	3
1.3 Learning Objectives.....	3
1.4 Scope.....	3
1.5 Methodologies	3
1.6 Sponsor.....	4
2 Technology.....	5
2.1 GeoIP.....	5
2.2 Service Oriented Architecture	6
2.3 Haversine Formula.....	7
2.4 Design Patterns.....	8
3 Presentation	9
3.1 Tools	9
3.2 Project Plan.....	10
3.3 Analysis phase.....	11
3.3.1 Analysis Phase problems and solutions.....	12
3.4 Design phase.....	13
3.4.1 Design Phase problems and solutions.....	15
3.5 Implementation phase	16
3.5.1 Implementation Phase problems and solutions	18
4 Evaluation	19
4.1 Software Requirements Analysis Document.....	19
4.2 Software Design Document	19
4.3 Implementation	20
4.4 Deployment	20
4.5 Quality of Deliverables.....	20
4.6 Other Deliverables.....	21

4.7 Commissioning Party Evaluation.....	21
5 Conclusion	22
6 Recommendations.....	24
Bibliography.....	25

Appendix 1 : Software Requirements Analysis Documentation

Appendix 2 : Software Design Documentation

Appendix 3 : Implementation Documentation

Vocabulary

On the go refers to a person who is constantly travelling from one place to another

Around refers to people who are in the vicinity of others

Abbreviations

UML	Unified Modeling Language
BIT	Business Information Technology
T-SQL	Transact Structured Query Language
GUI	Graphical User Interface
ORM	Object Relational Mapping
IP	Internet Protocol
MIT	Massachusetts Institute of Technology
OSI	Open Source Initiative
API	Application Programming Interface

1 Introduction

1.1 Background

The Internet is full of social networks and they keep growing day by day. Each social network has its own concept and focuses on certain audience. What actually defines a social network?

We define social network sites as web-based services that allow individuals to (1) construct a public or semi-public profile within a bounded system, (2) articulate a list of other users with whom they share a connection, and (3) view and traverse their list of connections and those made by others within the system. The nature and nomenclature of these connections may vary from site to site. (Ellison & Boyd 2007. Article 11.)

The idea of the thesis is based on my own experience as a traveler and as a spontaneous person. Unlike existing services in the market such as *CouchSurfing.com*, which requires the traveler to plan in advance who he is going to meet and when, the target of this project is to enable travelers to spontaneously meet fellow travelers around their current location as they are on the go without the need to stick to a certain plan.

TravelersAround is an online social network for travelers around the world. It provides travelers the facilities to interact with fellow travelers who are currently at their vicinity. The location of each traveler is determined when the traveler is logged on to the system and then a list of fellow online travelers around his current location is generated. The traveler can then communicate with other travelers and from that point and on, the sky is the limit.

1.2 Deliverables

The objectives of this thesis project were to develop the system through its *Software Development Life Cycle* as explained in *System Analysis & Design* (Dennis, Wixom & Roth 2006, 2). That includes the analysis phase, the design phase and the implementation and deployment of the system. The following are the main documents produced during this thesis:

- Software Requirements Analysis Documentation
- Software Design Documentation
- Software Implementation Documentation

1.3 Learning Objectives

Prior to beginning the project, few learning goals were set:

- Location-based programming
- Improve graphic designing and user interface styling
- Improve UML diagram skills
- Improve project management skills
- Develop a platform independent compatible application

1.4 Scope

This project covers the analysis, design and implementation phases of the architecture and main features of the system. The results consist of documentation and the system itself as a package of web application, web services and database.

Due to time limitation, the management panel of the system was left out.

1.5 Methodologies

The analysis and design phases of this project followed the *Waterfall Development Methodology*, which defines a sequential process for analyzing requirements of a system and

producing the software requirements analysis document and the software design document (Dennis et al. 2006, 10-11). The documentation produced followed the guidelines given in Haaga-Helia's course: Information System Development Project (SYS1TF080-7). I chose to use this methodology in order to produce a complete and thorough documentation of the system as required by the BIT bachelor thesis guidelines for product based thesis.

The implementation phase of this project followed the *Agile with SCRUM* development methodology, which emphasizes simple and iterative application development, for the implementation management part of the project (Dennis et al. 2006. 16-18). The decision to use this methodology in the implementation part comes from my own interest for experiencing this methodology as it is widely used nowadays, mostly in the implementation phase, by many companies around the world.

1.6 Sponsor

Avi Tours is a small tourism agency located in Israel. The owner of this company and the writer of the thesis came up with the system idea and saw it as a potential for future travelling portal. One of the key decision makers made by the company was that the system is developed free of charge and the business potential within.

2 Technology

In this chapter, the architecture and algorithm theory that were used and followed in the implementation of the system are described.

2.1 GeoIP

GeoIP is a proprietary technology developed by **MaxMind** that utilizes geographical position of a device that's connected to the Internet by its IP address (MaxMind 2011, What is GeoIP). The product is available for a certain fee as a web service or a database file, but is also available for as a free edition database file with lower resolving accuracy. MaxMind indicates that the free edition database file provides accuracy of up to 79% to a city level in the United States, while its commercial solutions can reach 83%.

MaxMind features an API which is compatible with common programming languages such as C#, Java, C/C++, etc. Developers use the API to query the database/web service to resolve an IP address. The results are then generated and are represented as a custom data type which contains the city, country, Internet provider and a geographical coordinates pair called latitude and longitude.

The *Geographical Coordinates System* defines a location on the surface of Earth by a set of numbers called Latitude and Longitude which are commonly used in navigation.

Lines of Latitude run parallel to Earth's equator and are divided into 180 equal portions from south to north. Lines of longitude run perpendicular to the equator and converge at the poles. (Riesterer, J. 2008. Geographical Coordinate System.)

TravelersAround uses MaxMind's C# API with its free edition database file to resolve an IP address into its Geographical Coordinates System representation. The information is then saved in the database for later calculations, using the *Haversine* formula (See section 2.3 **Haversine Formula**), to determine the closeness of travelers in the system.

2.2 Service Oriented Architecture

Service Oriented Architecture (SOA) is a software developing approach to separate concerns of an application design into distinct components which communicate with each other, yet act as an independent part of the application (Erl 2009. 32).

SOA is commonly used in a distributed business systems and allows platform independency when it comes to the clients consuming them. The service is described by a service contract (an interface), which specifies the functions offered by the service. Each function in the service acts as a separate autonomous unit that performs a certain task. The building block of SOA dictates that each service function should be completely independent and therefore not requiring its user to follow a certain workflow to run it. (Millett. 2010 .154-157.)

The development of TravelersAround followed the concept of SOA to enable ease of integration with possibly other existing systems in the market and to allow the development of different kind of user interfaces to be implemented easily (Millett. 2010 .154-157). Since the core of the system is contained within a web service, the client application can solely concentrate on building a well designed user interface and not worry about application logic. In addition, this enables smooth and easy future system integration with 3rd party systems.

2.3 Haversine Formula

Haversine formula is an equation to calculate distance between two points on a sphere by providing *longitude* and *latitude* coordinates and is normally used in navigation (Venness 2010. Distance).

$$\text{haversion} \left(\frac{d}{R} \right) = \text{haversion}(\varphi_2 - \varphi_1) + \cos(\varphi_1) \cos(\varphi_2) \text{haversion}(\Delta\lambda).$$

Figure 2-1: Haversine formula

- *haversion* is the Haversine function, $\text{haversion}(\theta) = \sin^2(\theta/2) = (1 - \cos(\theta))/2$
- *d* is the distance between the two points
- *R* is the radius of the sphere
- φ_1 is the latitude of point 1
- φ_2 is the latitude of point 2
- $\Delta\lambda$ is the longitude separation

The Haversine formula is used in the system to calculate the distances among travelers. The formula is represented as a T-SQL query in which the maximum radius of 20km is defined in order to generate a list of travelers around each other.

2.4 Design Patterns

A common mistake made by junior developers is to confront a problem which has already been solved by many developers in the past. Design Patterns come to ease their learning path through experiences and skills which were earned by senior developers.

Design Patterns are high-level abstract solution templates. Unlike a framework that can be simply applied to an application; design patterns are reached by refactoring application code and generalization of problems. Design Patterns originated from the experience and knowledge of programmers over many years. (Millett 2010. 4-12.)

Patterns are essential in application development as they indicate an intension through a common vocabulary when solving a problem at the design phase as well as at the implementation phase. They provide a description to solutions of complex problems and are language agnostic, which enables portability among different programming languages. (Millett 2010. 4-12.)

The key point of Design Patterns is that they have been tried and tested. Therefore, they are proven to be effective in the case they are aimed to solve. Developers who are familiar with Design Patterns have more chances to deliver their intensions to other developers through an application source code, when working in a team. (Millett 2010. 4-12.)

TravelersAround makes a wide use of Design Patterns to provide clean and maintainable code and provide interoperability between different layers. Among the used patterns are the *Repository Pattern*, *Domain Model* and *Factory Pattern* which are described in appendix 2: Software Design Documentation.

3 Presentation

This chapter describes the thesis project process. It provides information concerning what tools have been used and the steps that were followed.

3.1 Tools

The system was developed in C# .NET. The following tools were used:

For analysis and design:

- Microsoft Visio 2007
- Microsoft Word 2007
- Microsoft Project 2007

For implementation and deployment:

- Visual Studio 2010 Ultimate Edition
- Microsoft SQL Server Management Studio Express 2008
- IIS 7.5 Express
- FireFox 7 with FireBug and Developer Tools plug-ins
- Fiddler

3.2 Project Plan

The development process was carried out according to the work plan which was made before starting the project. The main tasks included:

Task	Outcomes
1. Analysis phase	Software Requirements Documentation
Defining Use Cases	General system specifications, Use Case Model
Specifying data to be stored	Class Models and Diagrams
Analyzing Use Case implementations and their information needs	Use Case Diagrams and Class Models
Usability Development Process	Description of the Usability Process and Form for the Usability Test
Finalizing documentation	Parameters and rules of functions
Review	Improved and corrected documentation
2. Design phase	Software Design documentation
User Interface Design	Page layouts and navigation, User Interface Style Guide
Planning the testing	Plan for the Software Testing
Database Design	Logical Database Design
Application Design	Software Architecture
Steering meeting	Improved and corrected documentation
3. Implementation phase	System Implementation & Documentation
Creating the Database	Database
Implementing User Interface	GUI
Implementing application code	Application code
Unit testing	Bug free code
Implementation phase complete	milestone reached
Planning the testing	Test Plan and Test Data
Testing activities	Testing minutes and documents

3.3 Analysis phase

The analysis phase outcome was the Software Requirements Documentation (Appendix 1). The foundation of the whole project and therefore was the first step in the system development process. The following are the descriptions of each step:

System Overview

The first step in the system development consisted of specifying what the system should do and what are the features and services it provides.

Use Cases

In this step the specifications turned to sketching and writing the proposed use cases on a paper. It was then reviewed and refined to form the systems specification and its users. After that, each specification was described to detail, resulting in the use case analysis model and diagram.

Class Model

In this step, the data to be stored by the use cases was defined. First, a sketch of a class diagram was made on a paper and later on refined to fit the specifications. The results of this step were the class model and diagram. Afterwards, the class model created was tested by creating an object life cycle diagram which considered an object from each class and its states.

Non-functional requirements

In this step, the non-functional requirements were determined. The usability requirements were specified along with the security and operational requirements.

Test Cases

In this step, the developed use cases were review through specific simulations of use cases scenarios. Each scenario is described in details along with a collaboration diagrams.

User Interface Classes

In this step, each use case was described as a user interface class. Specifying the data required and its types, the operations it provides and its associations with other user interface classes.

3.3.1 Analysis Phase problems and solutions

The following are problems which were encountered during this phase and their solutions:

Problem

The idea and main principles of this document were already known, but still some specific points were missing to complete the task.

Solution

Consulting the thesis supervisor provided a lot of answers and directions concerning which standards to follow.

Problem

I was unsure of the diagrams quality I produced as a result of lack of experience with UML notation.

Solution

Researching and studying UML specifications.

3.4 Design phase

The design phase main outcome was the Software Design Documentation (Appendix 2) based on the previous documentation produced in the analysis phase, the software requirements documentation. The work on this document couldn't be started until the previous document was finalized and clear understanding of what needs to be done had been reached. The design documentation served as the building block for the software implementation, providing clear tasks how and what should be done.

The following are the descriptions of each step:

System Specifications

In this step, the system technology to be used along with tools and frameworks has been decided and described.

Architecture Design

In this step, the system architectural design was decided and described to details. The architecture defined the model that will be used to develop the system along with patterns that will be followed to produce high quality system architecture.

User Interface Design

In this step, the user interface structure and navigation was designed. It specified the flow between each feature of the system and the steps needs to be done to reach it. Interface Metaphor, Interface Objects and Interface Actions were defined and described. Later on, user interface prototypes have been made according to the structure diagram that was produced earlier and according to the use cases analysis in the previous phase.

Application Design

In this step, the application design was decided and described. Each component was given a brief description and its part in the whole system. Later on, a sequence diagram was built to show how these components interact with each other to form a response for one use case and serve a user's request.

Database Design

In this step, the logical database structure was built and described, which formed the Logical Database Diagram. The diagram shows the relation among the database entities in a closer look to how it will be implemented in the system. The actual requirements of the database preferences were taken into consideration and a decision regarding which database system to use was made. In addition, each entity was given a detail description to form the Data Dictionary.

Test Plan

In this step, a test plan was made to cover the testing part of the implementation phase. Each test was given a brief description and its role in the implementation phase. Each test covered a specific scenario and was described in a test form, which then was used by the thesis supervisor to test the system.

3.4.1 Design Phase problems and solutions

The following are problems which were encountered during this phase and their solutions:

Problem

Lack of experience and knowledge with user interface prototyping.

Solution

Having a look at other social networks and some common user interface guidelines helped.

3.5 Implementation phase

In this phase, the system itself was developed according to the documents produced so far. The implementation proceeded fluently and rapidly, fulfilling the time limitation. The Implementation Documentation (Appendix 3) was produced to cover the main principles of the software code algorithms and architecture. The main deliverables of this phase were the system's source code and assets, which were released under the MIT OSI license, can be found here: <https://github.com/Nimrodda/TravelersAround>.

The following were the main steps and in this order:

Visual Studio Solution Creation

This was the first step in the implementation phase. All the necessary project types were added to the solution.

Database creation

In this step, the physical database diagram was designed in Visual Studio 2010 according to the logical database diagram design. Primary and foreign keys were defined along with their constraints.

Generating database entities with Entity Framework ORM

In this step, an Entity Framework file was added to the data access layer project, which then generated the database entities.

Creating the Data Access layer – the repository

In this step, the data access program class was built. This class included the basic CRUD operation to be done in the database.

Unit Testing the Data Access layer

In this step, the data access layer program classes were unit tested using MSTest, the built-in Visual Studio 2010 testing framework.

Creating the Application Logic layer – the service

In this step, the application logic program class was built. The class was designed as a WCF service contract, which included the system features presented as a web service.

Unit Testing the Application Logic layer

In this step, the application logic program class was unit tested using MSTest, the built-in testing framework in Visual Studio 2010.

Integration Testing for the two layers

In this step, the above layers were integrated together and tested.

Creating the Presentation layer – View Models, Controllers and Views (GUI)

In this step, the user interface was designed with Photoshop and then later on styled to fit HTML representation. Each user interface prototype was implemented in the system with its View Model to contain its data. The controllers were built to form the navigation and flow of the user interface.

Unit Testing the Presentation Layer

In this step, the application logic program class was unit tested using MSTest, the built-in testing framework in Visual Studio 2010.

Integration Testing all layers together

In this step, all layers were tested together.

Final touch ups and testing

In this step, the application was tested as a whole in black box testing.

3.5.1 Implementation Phase problems and solutions

No notable problems were found in this phase.

4 Evaluation

This part of the thesis report presents the actual results achieved during the thesis project and their value. The deliverables are the Software Requirements Analysis document, Software Design document, Implementation Document and the implemented system source code.

4.1 Software Requirements Analysis Document

The requirements analysis was done thoroughly and as expected, according to the plan and proposal. In this phase, Haaga-Helia study material definitely helped me complete the task and provided me with clear goals of what should be done and what is actually to be included in this document. The main topics of this document covered:

- The system overview
- Use case analysis
- Class model
- Non-functional requirements
- Test cases

4.2 Software Design Document

The design document was also done thoroughly and complete. Also in this part, I heavily relied on Haaga-Helia study material and guidelines to complete the task. The results are very sufficient and informative for a clear implementation of the system. This document was tightly written according to the software requirements analysis to maintain the consistency of the system.

The main topics of this document covered:

- System Specification
- Architecture Design
- User Interface Design
- Application Design
- Database Design
- Test Plan

4.3 Implementation

The implementation process followed the design document and the project plan very well. The standards that were set included designed patterns and models were also achieved. The requirement to make the system platform independent has been achieved. The system itself functions as it should be and covers all the required features and functions. The user interface design can definitely be improved, but this is due to lack of experience in GUI designing. The deliverable included the system itself as a Visual Studio 2010 solution folder along with a short implementation document, which gave a brief description of the system components.

4.4 Deployment

The deployment phase was postponed as a result of not finding suitable hosting provider. The sponsor and the developer are looking for offers and will deploy the system once a proper deal had been found. The service will be hosted under the domain: www.travelersaround.com.

4.5 Quality of Deliverables

The documents produced in this thesis followed the guidelines of the course "Information System Development Project" of Haaga-Helia. Each document has reached the required quality of that course.

The system implementation itself followed my own work experience and ethics as a commercial software developer for the past two years along with programming design patterns literature obtained from the Internet and textbooks. (See **Bibliography** for complete list of used literature).

4.6 Other Deliverables

The thesis project followed the guidelines for BIT bachelor's thesis in Haaga-Helia and therefore includes the additional deliverables which are part of the administration folder: Project Plan document, Final Report document, the agenda and steering meeting review documents, Progress reports.

4.7 Commissioning Party Evaluation

The results of the thesis were presented to the sponsor and accepted. The sponsor was satisfied with the results and agreed to release the service to the public. The sponsor and developer will look for an affordable web hosting company and deploy the application within the next couple of weeks after concluding this thesis.

5 Conclusion

The thesis project goal was to analyze, design and implement an online service for travelers around the world. The service behaves as a social network and enable travelers to see other travelers in their vicinity in real-time. During the thesis project's process, a number of documents have been produced which were the foundation of the implemented system. Among these documents were: Software Requirements Analysis documentation, Software Design documentation and a short Implementation documentation.

The chosen development methodology for the documentation part, *Waterfall Development*, was definitely a wise decision as it covered every aspect of the system according to the requirements of Haage-Helia's system based thesis. In addition, there was no business pressure to release the system . On the other hand, if the system wasn't developed as a thesis work, a lighter and a more iterative approach towards the documentation phase would have been used.

The chosen methodology for the implementation phase, *Agile development with SCRUM*, might not have been the best methodology for a solo developer. The reason for choosing this methodology was to experience the usage of this methodology in project implementation management. Nevertheless, I am confident that the chosen methodology proved to be very efficient for the short span system development such as in this thesis.

Apart from the documentation, there have been also learning points which were set and achieved during the thesis. The main learning objectives were: location-based programming, improving graphic designing, improving project management skills and experience project management using Agile with Scrum methodology.

The achieved deliverables fulfilled the requirements and allowed for smooth implementation of the system. The sponsor and the developer are very satisfied with the outcome of the implementation and see the system as a potential for future travelers portal. The system will surely contribute travelers around the world to get more out of

their trips; by meeting other travelers, they can discover things which they didn't plan to do and experience.

The service will be launched shortly after finalizing the thesis and will be provided free of charge.

6 Recommendations

The system development will continue after finishing this thesis project. As mentioned before, few features were left out of this project's scope as a result of time limitation. The sponsor and the developer have already discussed the possibility of mobile application development for the common smart phones in the market. In addition, a management panel will be developed in high priority.

Other possible future development might include integration with existing social networks, such as *Facebook*, *Twitter*, etc. More features will be added to the system to give it a more social network feeling like. Few of these features are planned to be: the ability of friends to comment of each other's profile, the ability to rate travelers' profiles, the ability to express their own experiences, the ability to define privacy rules, etc.

Bibliography

Boyd, d. m., & Ellison, N. B. 2007. Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, 13(1), article 11.

URL:<http://jcmc.indiana.edu/vol13/issue1/boyd.ellison.html>. Quoted: 8.10.2011

Dennis, A., Wixom, B.H. & Roth, R.M. 2006. System Analysis & Design. 3rd Edition. John Wiley & Sons, Inc.

MaxMind's IP Intelligence Solution

URL: <http://www.maxmind.com/app/ip-locate> Quoted: 8.10.2011

Erl, T. 2009. Service-Oriented Architecture (SOA): Concepts, Technology, and Design. Pearson Education, Inc. Crawfordsville, Indiana. United States.

Riesterer, J. 2008. Geographical Coordinate System.

URL: http://geology.isu.edu/geostac/Field_Exercise/topomaps/grid_assign.htm.

Quoted: 8.10.2011

Millett, S. 2010. Professional ASP.NET Design Patterns.

Wiley Publishing, Inc., Indianapolis, Indiana, USA.





Course Materials, Information System Development Project. SYS1TF080-7.

2011. Haaga-Helia. Helsinki.

Veness, C. 2010. Calculate distance, bearing and more between Latitude/Longitude points. URL: <http://www.movable-type.co.uk/scripts/latlong.html>. Quoted: 20.9.2011

Software Requirements Analysis Documentation

TravelersAround

Version	1.0		
Created by	Nimrod Dayan	18.8.2011	
Reviewed by	Juhani Välimäki	31.8.2011	
Approved by	Juhani Välimäki	31.8.2011	

Version	Date	Description	Author
0.1	18.8.2011	Introduction, system overview, use case diagram, services and actors have been added.	Nimrod Dayan
0.2	19.8.2011	Use case dependency diagram, use cases descriptions and data requirements have been added.	Nimrod Dayan
0.3	20.8.2011	Class model along with class diagram and class description, object life cycle have been added.	Nimrod Dayan
0.4	21.8.2011	Summary of data use and non-functional requirements have been added.	Nimrod Dayan
0.5	22.8.2011	Test cases, scenarios and collaboration diagrams have been added.	Nimrod Dayan
0.6	23.8.2011	User interface class's description has been added.	Nimrod Dayan
1.0	31.8.2011	Corrections after review of steering meeting.	Nimrod Dayan

Table of contents

1	Introduction.....	1
2	System overview.....	1
2.1	Overall workflow description.....	1
2.2	Use Case diagram.....	2
2.3	Actors.....	2
2.4	Services.....	3
2.4.1	Register.....	3
2.4.2	Log on.....	3
2.4.3	Search for fellow travelers.....	3
2.4.4	Update profile.....	3
2.4.5	Manage friends list.....	3
2.4.6	Manage messages.....	4
3	Use cases.....	5
3.1	Sub use cases description.....	5
3.1.1	Send message.....	5
3.1.2	Read message.....	5
3.1.3	Add friend.....	5
3.2	Use case analysis.....	6
3.2.1	Register.....	6
3.2.2	Logon.....	7
3.2.3	Search for fellow travelers.....	8
3.2.4	Add friend.....	8
3.2.5	Update profile.....	9
3.2.6	Manage friends list.....	10
3.2.7	Manage messages.....	11
3.2.8	Send message.....	12
3.2.9	Read message.....	13
3.3	Data requirements.....	13
3.3.1	Register.....	13
3.3.2	Logon.....	13

3.3.3	Search for travelers.....	14
3.3.4	Add friend	14
3.3.5	Update profile	14
3.3.6	Manage friends list.....	14
3.3.7	Send message	15
3.3.8	Read message	15
3.3.9	Manage messages.....	15
4	Class model.....	16
4.1	Conceptual Level Class diagram.....	16
4.2	Class definitions.....	17
4.2.1	Traveler	17
4.2.2	Message.....	18
4.3	Objects life cycle.....	19
4.3.1	Traveler	19
4.3.2	Message.....	20
5	Summary of data use.....	21
6	Non-functional requirements	22
6.1	Usability	22
6.1.1	Navigation	22
6.1.2	Efficient usage.....	22
6.1.3	Error handling and messages.....	22
6.2	Security	23
6.3	Performance.....	24
6.4	Operational requirements.....	24
6.4.1	Traveler’s location determination.....	24
6.4.2	Platform independency.....	24
7	Test Cases.....	25
7.1	Register	25
7.1.1	Scenario.....	25
7.1.2	Collaboration Diagram	25
7.2	Logon.....	26
7.2.1	Scenario.....	26

7.2.2	Collaboration Diagram	26
7.3	Search for fellow travelers.....	27
7.3.1	Scenario.....	27
7.3.2	Collaboration Diagram	27
7.4	Update profile.....	28
7.4.1	Scenario.....	28
7.4.2	Collaboration Diagram	28
7.5	Manage friends list	29
7.5.1	Scenario.....	29
7.5.2	Collaboration Diagram	29
7.6	Manage messages.....	30
7.6.1	Scenario.....	30
7.6.2	Collaboration Diagram	30
7.7	Add friend	31
7.7.1	Scenario.....	31
7.7.2	Collaboration Diagram	31
References	32
Appendix 1 – User Interface classes	33
Register	33
Logon	33
Search for travelers	34
Update profile	35
Manage friends list	36
Send message	36
Read message	37
Manage messages	37
Add friend	38

1 Introduction

The purpose of this document is to analyze and describe the requirements for my thesis project, TravelersAround, and establish the foundation for the next step in the development of the system: the design phase. It is based on the system proposal by the sponsor and describes theoretically what features the system will have, who are its users and what are the main benefits it provides.

The system services and its class models are presented in UML notation along with detailed description. The documentation contents follow the guidelines taught in Haa-ga-Helia for Software Requirements Analysis document.

2 System overview

TravelersAround is an online service provided to travelers around the world, who would like to spontaneously meet fellow travelers or even the locals, without any need to plan anything in advance.

2.1 Overall workflow description

New travelers register to the system and fill in their personal details. After successfully completing the registration phase, the system will identify their location or feed in their location manually.

Once the location is determined, the traveler can search for fellow online travelers in his area and receive a list of the results. He can then add the person to his friends list or just send a private message. From this point on, it's all up to the travelers how they decide to continue.

2.2 Use Case diagram

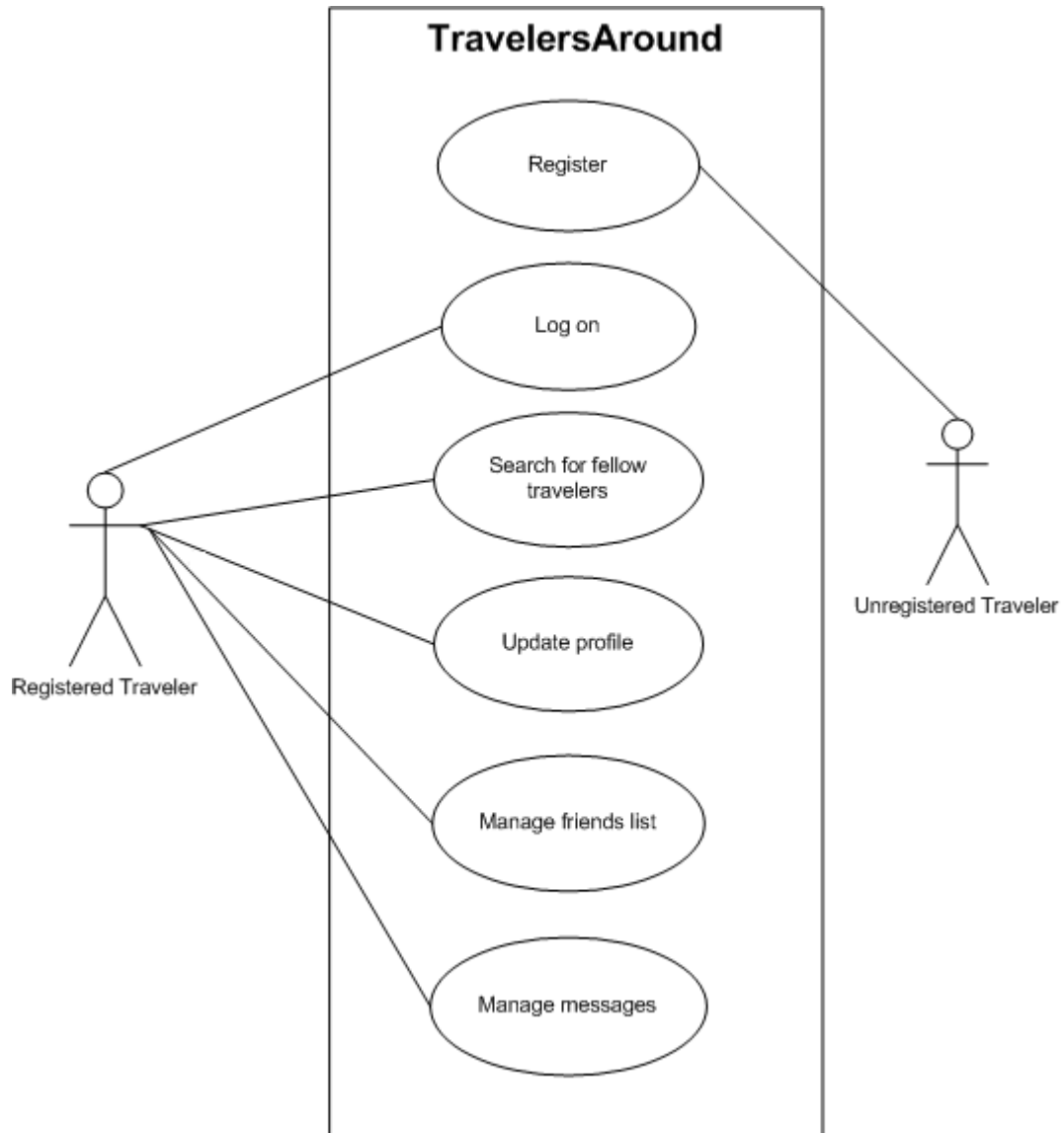


Figure 2-1: The system's Use Case diagram- the main features

2.3 Actors

Unregistered Traveler

A random traveler who's not registered and therefore has no access to the system's features besides registration.

Registered Traveler

The registered traveler is the main user of the system and can use all the features and functions in the system without restrictions.

2.4 Services

The following is a brief description of each use case presented in the figure 2-1:

2.4.1 Register

This use case describes how a new traveler becomes a member of the system. The traveler enters his personal details into the form. As a result, the system creates a user profile and the user gets the privileges required to interact with the system.

2.4.2 Log on

This use case describes how an existing member identifies himself and gains the required privileges to interact with the system.

2.4.3 Search for fellow travelers

This use case describes how a traveler can search for nearby fellow travelers. The search is defined according to the user's current location and a filter, which is defined by the traveler, specifying whether to show only available or unavailable marked persons in the search results. The results are displayed in a list providing names, profile pictures, gender, age, status messages and availability mark of fellow travelers nearby. The traveler can then send a message, initiate a chat or add a person to his friends list.

2.4.4 Update profile

This use case describes how the traveler can update his personal details, profile picture, status message and availability mark.

2.4.5 Manage friends list

This use case describes how the traveler can manage his friends list and remove friends from the list by setting a mark next to each friend's name.

2.4.6 Manage messages

This use case describes how the traveler can browse his messages list and remove unwanted messages. Messages which are already read are marked as read. The user can reply or send a new message (by extension use cases). The user can also send a new message, but in this use case the user is restricted to send new messages **only** to his friends. When sending a new message, the user chooses the recipient from his friends list.

3 Use cases

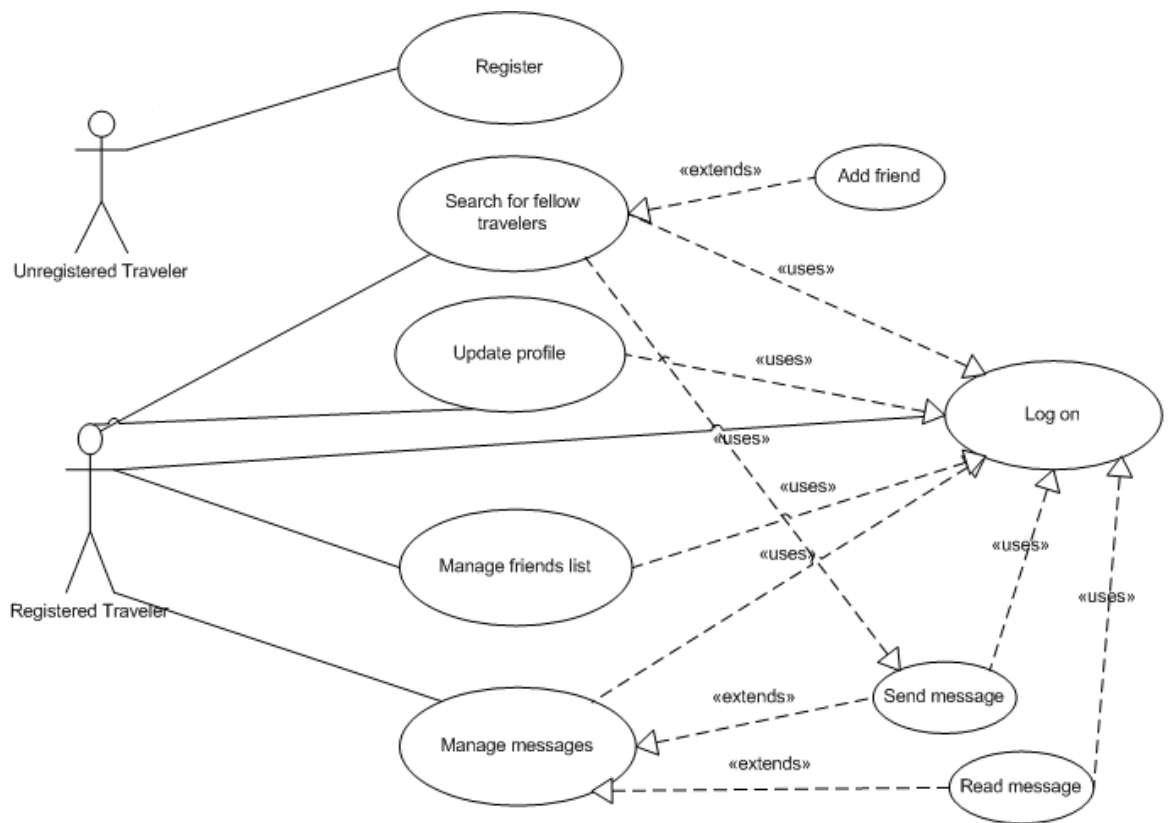


Figure 3-1: Use case dependency diagram

The above diagram shows the main use cases with sub uses cases and their relation.

3.1 Sub use cases description

3.1.1 Send message

This use case describes how traveler can send a message.

3.1.2 Read message

This use case describes how traveler can read a message.

3.1.3 Add friend

This use case describes how traveler adds a fellow traveler, whom he found in the search results for nearby fellow travelers, to his friends list.

3.2 Use case analysis

The following are detailed descriptions of each use case. The following notation is used:

Step marked with the letter E represents a possible error on a specific step.

Step marked with the letters RE represents the detailed error message.

Step marked with the letter V represents another variation the use case may take.

3.2.1 Register

Actor	Unregistered Traveler
Pre condition	-
Goal	A new traveler is stored in the system and given a unique identifier
Step	Description
1	The actor enters his details (name, email address, password, birth date, gender) and accept the terms of use
2	The actor defines that the information has been entered and the system should save it
3	The system grants access to interact with itself
E3a	The actor did not enter all the required information or entered invalid data (RE3)
3.1	The system displays an error message
3.2	The system informs the user about which information is missing/invalid
3.3	The actor indicates that they have recognized the message
3.4	The system redirects the actor back to step 1
RE3	Required information: all information is required Invalid data: data that has a length not allowed by the system, an incorrect type or an existing email address in the system

3.2.2 Logon

Actor	Registered Traveler
Pre condition	Actor is already registered in the system
Goal	Identify and authenticate the traveler and grant permissions
Step	Description
1	The actor enters his credentials (email address, password)
2	The actor defines that the information has been entered and the system should process it
3	The system validates credentials and grants access to interact with itself
E3a	The actor did not enter all the required information or entered invalid data (RE3)
3.1	The system displays an error message
3.2	The system informs the user about which information is missing/invalid
3.3	The actor indicates that they have recognized the message
3.4	The system redirects the actor back to step 1
RE3	<p>Required information: all information is required</p> <p>Invalid data: data that has a length not allowed by the system, an incorrect type or format and incorrect email/password</p>

3.2.3 Search for fellow travelers

Actor	Registered Traveler
Pre condition	Actor is logged onto system
Goal	Search for fellow travelers around the actor according to his location
Step	Description
1	The actor chooses whether he wants to see all nearby travelers or only available ones
2	The actor defines that the information has been entered and the system should process it
3	The system processes the request and shows the results of the search as a list
4	The actor sends a message to selected traveler from the results (Include UC: Send message)
V2a	Add friend
2.1	The actor clicks on the add friend button, which is next to each person displayed in the search results' list (Extension UC: Add friend)

3.2.4 Add friend

Actor	Registered Traveler
Pre condition	Actor is logged onto system
Goal	Friend is added to the traveler's friends list
Step	Description
1	The actor defines that he wants to add the person to his friends list and the system should save the changes
2	The system processes the request and shows a message indicating that the person has been added to the friends list
E2	The actor tried to add a person who's already in his friends list
2.1	The system displays an error message
2.2	The system informs the user about which information is missing/invalid

2.3	The actor indicates that they have recognized the message
2.4	The system redirects the actor back to step 1

3.2.5 Update profile

Actor	Registered Traveler
Pre condition	Actor is logged onto system
Goal	The traveler information is updated
Step	Description
1	The system displays the traveler's information (name, email address, password, birth date, gender, status message, availability mark)
2	The actor modifies the desired information
3	The actor defines that the information has been updated and the system should save it
4	The system displays a message that the save was successful
V2a	Deletion Mark
2.1	The actor adds a deletion mark to the profile
E4	The actor removed required information or entered invalid data (RE4)
4.1	The system displays an error message
4.2	The system informs the user about which information is missing/invalid
4.3	The actor indicates that they have recognized the message
4.4	The system redirects the actor back to step 2
RE4	Required information: all information is required Invalid data: data that has a length not allowed by the system, an incorrect type or an existing email address in the system

3.2.6 Manage friends list

Actor	Registered Traveler
Pre condition	Actor is logged onto system
Goal	The traveler's friends list is modified
Step	Description
1	The system displays the actor's friends in a list with a checkbox next to each name
2	The actor checks the box next to the friends he wishes to remove from the list
3	The actor defines that the system should save it
4	The system displays a message that the save was successful
E4	The actor hasn't indicated which friend he wishes to remove (RE4)
4.1	The system displays an error message
4.3	The actor indicates that they have recognized the message
4.4	The system redirects the actor back to step 2
RE4	Required information: at least one checkbox has to be selected

3.2.7 Manage messages

Actor	Registered Traveler
Pre condition	Actor is logged onto system
Goal	The traveler's messages are managed
Step	Description
1	The system displays traveler's messages in a list. Each message has its sender's name, subject, date and state mark, which defines if the message was read or not
2	The actor clicks on the message's subject to read it
3	The system displays the message content (Extension UC: Read message)
V2a	Deletion mark
2.1	The actor adds a deletion mark to a message in the list by marking the checkbox next to the message he wishes to delete
2.2	The actor defines that the system should save it
2.3	The system processes the request and redirects the actor to step 1
V2b	Send message
2.1	The actor can click on <i>send message</i> button to send new message (Extension UC: Send message)
E2.3	The actor hasn't indicated which message he wishes to remove (RE2.3)
2.3.1	The system displays an error message
2.3.2	The actor indicates that they have recognized the message
2.3.3	The system redirects the actor back to step 1
RE2.3	Required information: at least one checkbox has to be selected

3.2.8 Send message

Actor	Registered Traveler
Pre condition	Actor is logged onto system
Goal	Traveler sends a new message
Step	Description
1	The actor writes the message (recipient name, subject, content)
2	The actor defines that the message has been written and the system should send it
3	The system processes the request and indicates the operation was successful
E3a	The actor did not enter all the required information or entered invalid data (RE3)
3.1	The system displays an error message
3.2	The system informs the user about which information is missing/invalid
3.3	The actor indicates that they have recognized the message
3.4	The system redirects the actor back to step 1
RE3	Required information: all information is required Invalid data: data that has a length not allowed by the system or an incorrect type

3.2.9 Read message

Actor	Registered Traveler
Pre condition	Actor is logged onto system
Goal	Traveler reads a message
Step	Description
1	The system displays the message (sender name, subject, content) and sets the message state as read
V2a	Reply message
2.1	The actor clicks the reply button
2.2	The system redirects the actor to UC: Send message and prepares the reply message (sets the new message's recipient to the original message's sender and inserts the subject & content of the original message to the subject & content of the new message)

3.3 Data requirements

The following is the data required by each use case in the conceptual level.

3.3.1 Register

Email address
Password
Name
Date of birth
Gender
Save message
Registration status message

3.3.2 Logon

Email address

Password

3.3.3 Search for travelers

Filter: Availability mark
Travelers list result including pictures, names and status messages
Send message button
Add to friends list button

3.3.4 Add friend

Traveler name
Update message

3.3.5 Update profile

Email address
Password
Name
Date of birth
Gender
Status message
Availability mark
Profile picture
Deletion mark
Update message

3.3.6 Manage friends list

Deletion mark
Friends list
Update message

3.3.7 Send message

Sender
Subject
Content
Recipient
Send message status

3.3.8 Read message

Message ID from messages list

3.3.9 Manage messages

Messages list including message ID, sender, subject and deletion mark
Delete message status

4 Class model

In this chapter I present the conceptual class model diagram followed by each class detailed description presented in a table.

4.1 Conceptual Level Class diagram

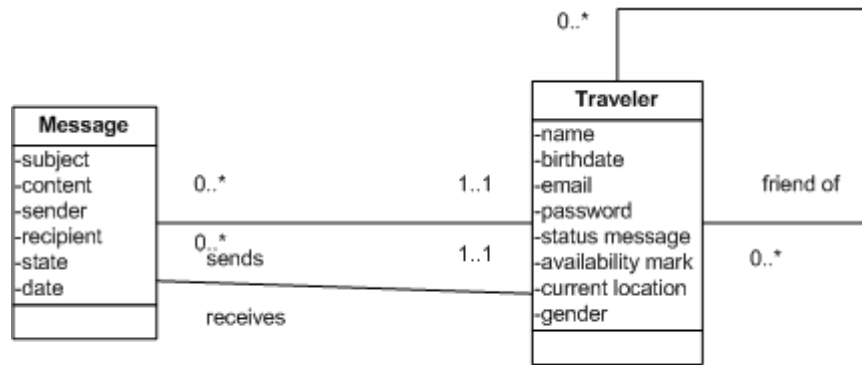


Figure 4-1: Conceptual Level Class diagram

Class descriptions are in the next page.

4.2 Class definitions

4.2.1 Traveler

Class name	Traveler		
Definition	Represents an individual traveler in the system		
Super class	-		
Attributes	email	Alphanumeric, 50 characters	Traveler's email address. Also used as the username to log onto the system
	Password	Alphanumeric, 8 characters	Traveler's password. Must be 6-8 characters and contain both digits and letters
	Name	Alphanumeric, 50 characters	Traveler's name
	Date of birth	Alphanumeric, 8 characters	Traveler's date of birth
	Gender	Alphanumeric, 6 characters	Traveler's gender
	Status message	Alphanumeric, 150 characters	Traveler's status message to be displayed on search results
	Availability mark	yes/no	Specifies whether the traveler is available or not at the moment
Associations	Traveler is friend of 0 to many travelers Traveler sends 0 to many messages Traveler receives 0 to many messages		
Operations	Create Update Delete		
Responsibilities	-		
Volume	-		

4.2.2 Message

Class name	Message		
Definition	Represents a single message managed by traveler in the system		
Super class	-		
Attributes	Sender	Alphanumeric, 50 characters	The traveler who sent the message
	Subject	Alphanumeric, 50 characters	Subject of message
	Content	Alphanumeric, 500 characters	Message content text
	Recipient	Alphanumeric, 50 characters	Receiver's name
	State	Yes/No	Yes if message was read
	Date	Date and time	Date message was sent
Associations	Message is managed by one to one traveler		
Operations	Create Update Delete Create link to two traveler (sender and receiver)		
Responsibilities	Message knows the traveler it is connected to		
Volume	-		

4.3 Objects life cycle

4.3.1 Traveler

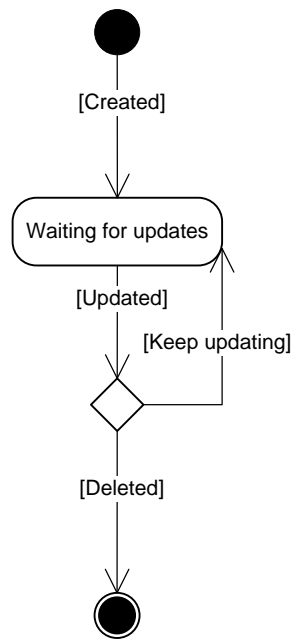


Figure 4-2: Traveler object state diagram

4.3.2 Message

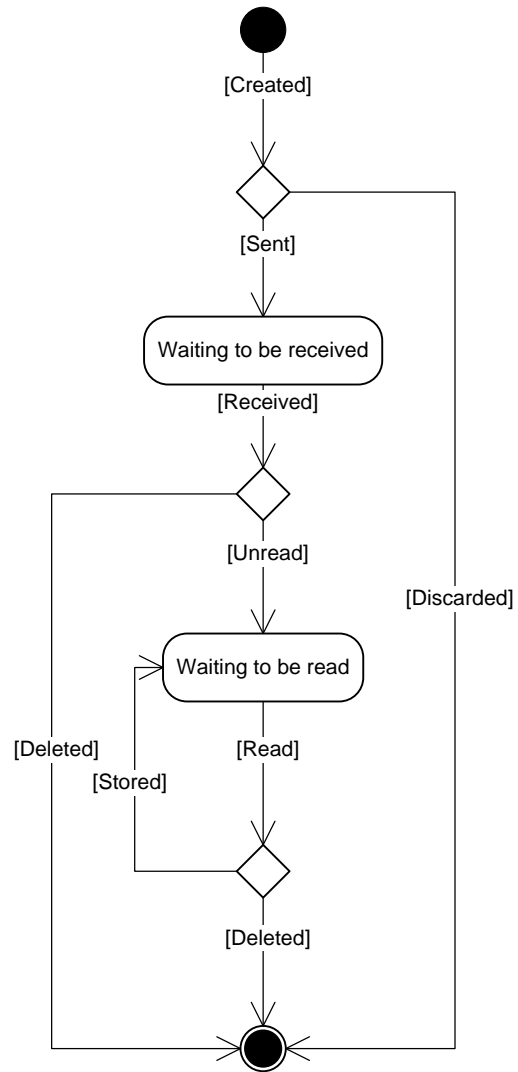


Figure 4-3: Message object state diagram

5 Summary of data use

In this chapter we can see the relation between each use case and the classes it uses.

Class	Traveler	Message
Use case		
Register	C	-
Logon	R	-
Update profile	RUD	-
Search for fellow travelers	R	-
Manage friends list	RU	-
Add friend	U	-
Manage message	-	RUD
Send message	-	C
Read message	-	R

C-Create, U-Update, R-Read, D-Delete

6 Non-functional requirements

6.1 Usability

6.1.1 Navigation

The system will feature easy navigation paths to its functions through simple to use panel, decorated with descriptive icons per each function. Only the main use cases will be displayed in the panel. Therefore, decreasing the amount of buttons on the screen, and prevent user's confusion. Each main use case will be accompanied with its sub-panel and its functions, if any sub use cases exist.

6.1.2 Efficient usage

The system will use a principle of maximizing usage of stored data, decreasing the need for typing. Users will choose from lists of stored information menus instead of manually typing it. This will increase the efficiency of the user and the results expected from the system.

6.1.3 Error handling and messages

The system will include descriptive-human friendly error messages along with suitable icons to guide the user exactly what went wrong and how to correct it.

6.2 Security

The system is membership based, meaning that each user has a unique username and password. The user is authenticated by the system and is then authorized to interact with the system. All members have the same access rights to the system, there are no administrators or super users at this point.

Actor	Registered Traveler	Unregistered Traveler
Use case		
Register	-	X
Logon	X	-
Update profile	X	-
Search for fellow travelers	X	-
Manage friends list	X	-
Add friend	X	-
Manage message	X	-
Send message	X	-
Read message	x	-

6.3 Performance

The system will provide data in real-time. Therefore, it is important that all features are optimized to provide current data in the fastest way possible without any delays. Each response shouldn't take than 10 seconds.

6.4 Operational requirements

6.4.1 Traveler's location determination

The system will draw information from a third party service, which contains information regarding location determination around the world.

6.4.2 Platform independency

The system will be developed in a way that features easy integration with different technologies on different platforms.

7 Test Cases

In this chapter, each use case and the appropriate class are tested in a specific test scenario in order to confirm the validity and consistency of the content analyzed so far in this document. The test is presented in a collaboration diagram, which shows what use cases and classes are used in the given scenario and what are the steps taken. The tables show the main steps from each use case scenario.

7.1 Register

A new traveler is registered to the system. All required fields are entered and valid. The traveler becomes a user of the system and gets permissions.

7.1.1 Scenario

Actor	Unregistered Traveler
Pre condition	-
Goal	A new traveler is stored in the system and given a unique identifier
3	The actor enters his details (name, email address, password, birth date, gender) and accept the terms of use
4	The actor defines that the information has been entered and the system should save it
6	The system grants access to interact with itself

7.1.2 Collaboration Diagram

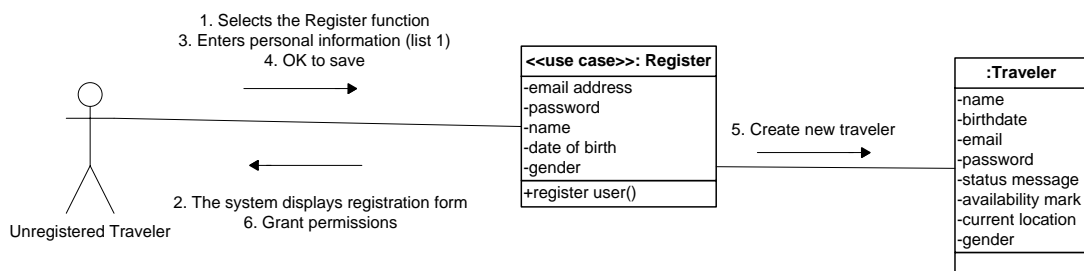


Figure 7-1: Register scenario collaboration diagram

List 1: name, email address, password, birth date, gender

7.2 Logon

A registered traveler is logging on to the system. All required fields are entered and valid. The system should grant permission to use itself.

7.2.1 Scenario

Actor	Registered Traveler
Pre condition	Actor is already registered in the system
Goal	Identify and authenticate the traveler and grant permissions
3	The actor enters his credentials (email address, password)
4	The actor defines that the information has been entered and the system should process it
6	The system validates credentials and grants access to interact with itself

7.2.2 Collaboration Diagram

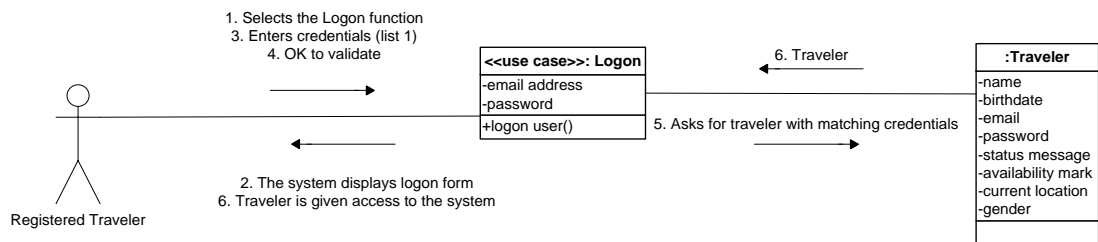


Figure 7-2: Logon scenario collaboration diagram

List 1: email address, password

7.3 Search for fellow travelers

Registered traveler makes a search for travelers around him. The result list shows a number of travelers in the list and then he sends a message to one of the travelers in the list. All required fields are entered and valid and the message is successfully sent.

7.3.1 Scenario

Actor	Registered Traveler
Pre condition	Actor is logged onto system
Goal	Search for fellow travelers around the actor according to his location
3	The actor chooses whether he wants to see all nearby travelers or only available ones
4	The actor defines that the information has been entered and the system should process it
6	The system processes the request and shows the results of the search as a list
7	The actor sends a message to selected traveler from the results (Include UC: Send message)

7.3.2 Collaboration Diagram

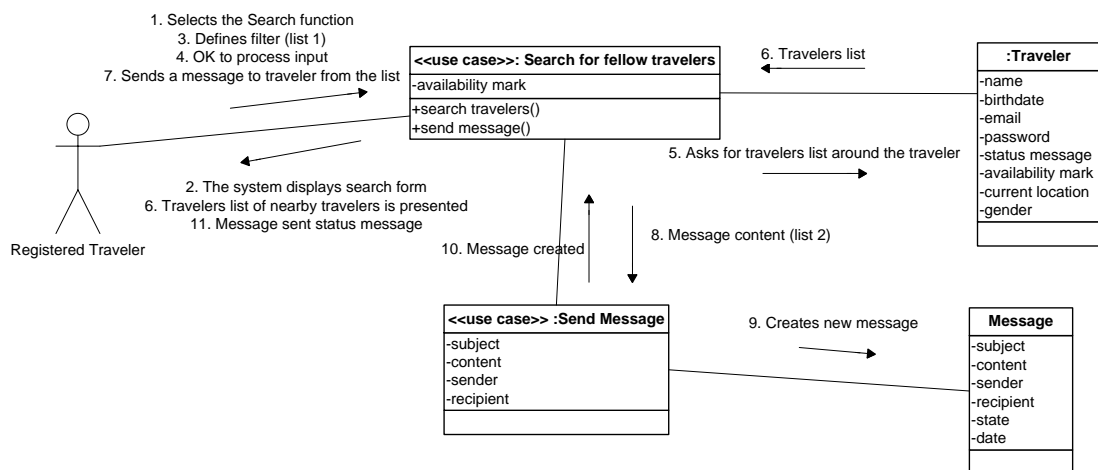


Figure 7-3: Search for fellow travelers scenario collaboration diagram

List 1: availability mark

List 2: subject, content, sender, recipient

7.4 Update profile

A registered traveler updates his profile information. The traveler's current information is displayed. All required fields are filled and valid. The system saves the data and indicates that it was successful.

7.4.1 Scenario

Actor	Registered Traveler
Pre condition	Actor is logged onto system
Goal	The traveler information is updated
2	The system displays the traveler's information (name, email address, password, birth date, gender, status message, availability mark)
3	The actor modifies the desired information
4	The actor defines that the information has been updated and the system should save it
6	The system displays a message that the save was successful

7.4.2 Collaboration Diagram

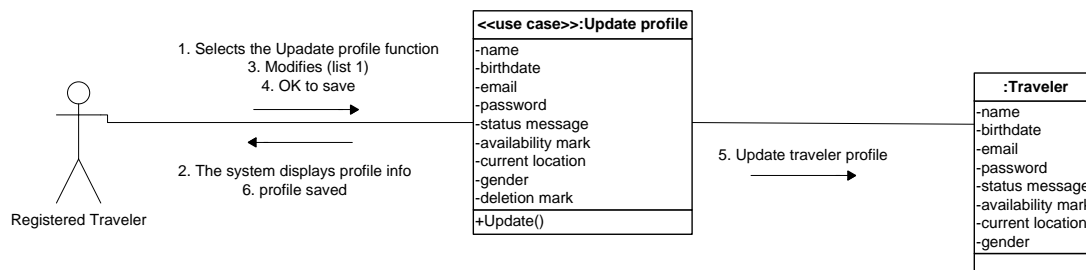


Figure 7-4: Update profile scenario collaboration diagram

List 1: name, email address, password, birth date, gender, status message, and availability mark

7.5 Manage friends list

A registered traveler who has a number of friends in his list and removes friends from his friends list by marking the checkbox next to each friend he wishes to remove and saves the list. The list is then refreshed and the friend is no longer there.

7.5.1 Scenario

Actor	Registered Traveler
Pre condition	Actor is logged onto system
Goal	The traveler's friends list is modified
2	The system displays the actor's friends in a list with a checkbox next to each name
3	The actor checks the box next to the friends he wishes to remove from the list
4	The actor defines that the system should save it
6	The system displays a message that the save was successful

7.5.2 Collaboration Diagram

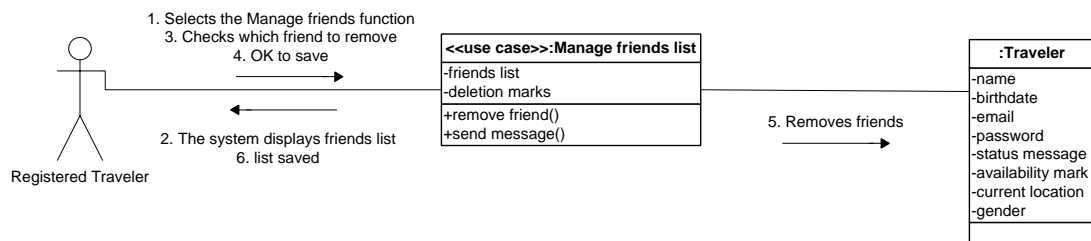


Figure 7-5: Manage friends' list scenario collaboration diagram

7.6 Manage messages

A registered traveler reads a message from his messages list which has a number of messages that he received by clicking on the message's subject. The message content should display.

7.6.1 Scenario

Actor	Registered Traveler
Pre condition	Actor is logged onto system
Goal	The traveler's messages are managed
4	The system displays traveler's messages in a list. Each message has its sender's name, subject, date and state mark, which defines if the message was read or not
5	The actor clicks on the message's subject to read it
9	The system displays the message content (Extension UC: Read message)

7.6.2 Collaboration Diagram

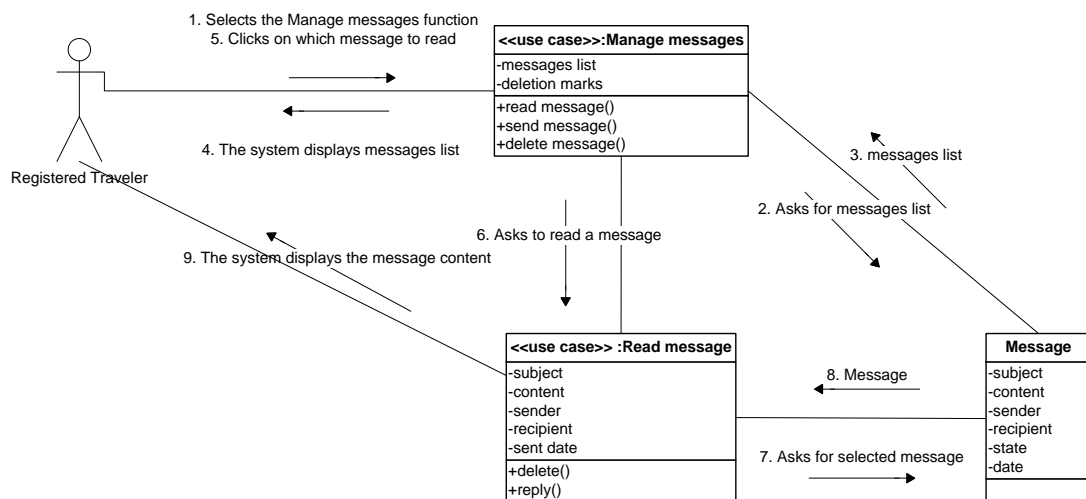


Figure 7-6: Manage messages scenario collaboration diagram

7.7 Add friend

The traveler makes a search for travelers around him. A results list of number of traveler is presented. The traveler adds one of the persons to his friends list by clicking on the add friend button. The process is successful and the traveler gets a notification of the process success.

7.7.1 Scenario

Actor	Registered Traveler
Pre condition	Actor is logged onto system
Goal	Search for fellow travelers around the actor according to his location
3	The actor chooses whether he wants to see all nearby travelers or only available ones
4	The actor defines that the information has been entered and the system should process it
6	The system processes the request and shows the results of the search as a list
7	The actor clicks on the add friend button, which is next to each person displayed in the search results' list (Extension UC: Add friend)

7.7.2 Collaboration Diagram

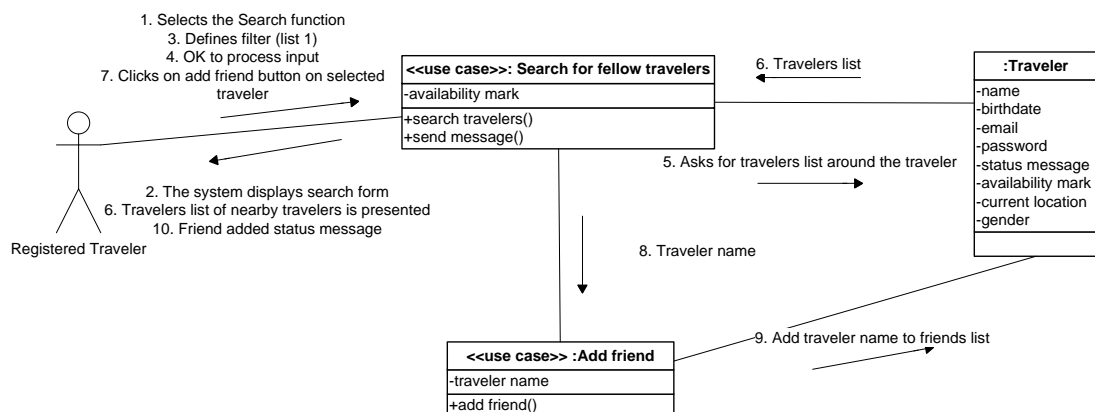


Figure 7-7: Add friend scenario collaboration diagram

References

Dennis, A., Wixom, B.H. & Roth, R.M. 2006. System Analysis & Design. 3rd Edition. John Wiley & Sons, Inc.

Course Materials, Information System Development Project. SYS1TF080-7. 2011. Haaga-Helia.

Tang, T.D., 2009. Thesis: Requirements Engineering Process for Sales Management System Case study: Tin Phong Trading Co., Ltd. Haaga-Helia. Helsinki.

Pukkila, V. 2009. Thesis: Hotel System with Java and MySQL. Haaga-Helia. Helsinki.

Appendix 1 – User Interface classes

In this appendix, each scenario from the Test Cases chapter is described as a class, showing its used data and operations.

Register

Class name	Register		
Definition	The program class taking care of registering new users in the system		
Super class	-		
Attributes	Email address	Alphanumeric, 50	Email address of traveler
	Password	Alphanumeric, 8	Password of traveler
	Name	Alphanumeric, 50	Name of traveler
	Date of birth	Alphanumeric, 8	Date of birth of traveler
	Gender	Alphanumeric, 6	Gender of traveler
Operations	Register user		
Associations	-		
Responsibilities	-		
Volume	-		

Logon

Class name	Logon		
Definition	The program class taking care of logging on existing users in the system		
Super class	-		
Attributes	Email address	Alphanumeric, 50	Email address of traveler
	Password	Alphanumeric, 8	Password of traveler
Operations	Logon user		
Associations	-		
Responsibilities	-		
Volume	-		

Search for travelers

Class name	Search		
Definition	The program class taking care of searching for online travelers around the traveler and communicate with them		
Super class	-		
Attributes	Availability mark	Boolean	Filter for search indicating whether to show only travelers who are available or show everybody
Operations	Search travelers Send message		
Associations	-		
Responsibilities	-		
Volume	-		

Update profile

Class name	Update profile		
Definition	The program class taking care of updating traveler personal profile in the system		
Super class	-		
Attributes	Email address	Alphanumeric, 50	Email address of traveler
	Password	Alphanumeric, 8	Password of traveler
	Name	Alphanumeric, 50	Name of traveler
	Date of birth	Alphanumeric, 8	Date of birth of traveler
	Gender	Alphanumeric, 6	Gender of traveler
	Status message	Alphanumeric, 150 characters	Free text for the traveler to write a message which will be displayed in the search results along with his picture and personal details
	Availability mark	Boolean	Indicates whether the traveler is available or not
	Deletion mark	Boolean	Indicates that the traveler wishes to remove himself from the system and stop using the service
Operations	Update		
Associations	-		
Responsibilities	-		
Volume	-		

Manage friends list

Class name	Manage friends list		
Definition	The program class taking care of managing traveler's friends list		
Super class	-		
Attributes	Friends list	List of Traveler objects	List of the traveler's friends
	Deletion marks	List of Boolean	List of marks indicating which friend the traveler wishes to remove from the list
Operations	Remove friend Send message		
Associations	-		
Responsibilities	-		
Volume	-		

Send message

Class name	Send message		
Definition	The program class taking care of sending a message in the system between two travelers		
Super class	-		
Attributes	Sender	Alphanumeric, 50 characters	The traveler who sent the message
	Subject	Alphanumeric, 50 characters	Subject of the message
	Content	Alphanumeric, 500 characters	Body content of the message
	Recipient	Alphanumeric, 50 characters	The traveler to receive the message
Operations	Send		
Associations	-		
Responsibilities	-		
Volume	-		

Read message

Class name	Read message		
Definition	The program class taking care of displaying a received message		
Super class	-		
Attributes	Sender	Alphanumeric, 50 characters	The traveler who sent the message
	Subject	Alphanumeric, 50 characters	Subject of the message
	Content	Alphanumeric, 500 characters	Body content of the message
	Recipient	Alphanumeric, 50 characters	The traveler to receive the message
	Sent date	Date and time	The date and time message was sent
Operations	Reply Delete		
Associations	-		
Responsibilities	-		
Volume	-		

Manage messages

Class name	Manage messages		
Definition	The program class taking care of managing traveler's messages		
Super class	-		
Attributes	messages list	List of Message objects	List of the traveler's messages
	Deletion marks	List of Boolean	List of marks indicating which message the traveler wishes to remove from the list
Operations	Read message Send message Delete message		
Associations	-		
Responsibilities	-		
Volume	-		

Add friend

Class name	Add friend		
Definition	The program class taking care of adding a friend to traveler's friends list		
Super class	-		
Attributes	Traveler name	Alphanumeric, 50	Name of traveler to be added to friends list
Operations	Add friend		
Associations	-		
Responsibilities	-		
Volume	-		



HAAGA-HELIA
University of Applied Sciences

Software Design Documentation

TravelersAround

Version	1.0	
Created by	Nimrod Dayan	23.8.2011
Reviewed by	Juhani Välimäki	31.8.2011
Approved by	Juhani Välimäki	31.8.2011

Version	Date	Description	Author
0.1	23.8.2011	System specifications has been added	Nimrod Dayan
0.2	24.8.2011	Architecture design has been added	Nimrod Dayan
0.3	25.8.2011	User interface diagram and user interface structure have been added	Nimrod Dayan
0.4	26.8.2011	User interface prototypes have been added	Nimrod Dayan
0.5	27.8.2011	Application design has been added	Nimrod Dayan
0.6	28.8.2011	Database design has been added	Nimrod Dayan
0.7	29.8.2011	Test plan has been added	Nimrod Dayan
1.0	31.8.2011	Corrections made after review.	Nimrod Dayan

Table of contents

Abbreviations.....	1
1 Introduction.....	2
2 System Specifications.....	2
2.1 Technical Environment.....	2
2.2 System Integration	2
2.3 Tools	2
2.4 Frameworks.....	3
3 Architecture Design	4
3.1 Multi-tier Architecture Model.....	4
3.2 Layers Description	5
3.2.1 Presentation layer	5
3.2.2 Application layer.....	5
3.2.3 Data Access layer.....	5
3.3 Design Patterns.....	5
4 User Interface Design.....	7
4.1 Interface Structure Diagram	7
4.2 Interface Standards Design.....	8
4.2.1 Interface Metaphor.....	8
4.2.2 Interface Objects	8
4.2.3 Interface Actions	8
4.2.4 Interface Icons	9
4.2.5 Input Design	9
4.2.6 Output Design	9
4.2.7 Interface Template	10
4.3 Interface Design Prototype.....	11
4.3.1 Logon and landing page	11
4.3.2 Registration.....	12
4.3.3 Update Profile.....	13
4.3.4 Search for travelers around.....	14
4.3.5 Manage friends list.....	15
4.3.6 Manage messages.....	16

4.3.7	Send message	17
4.3.8	Read message	18
5	Application Design	19
5.1	Components Description.....	19
5.1.1	View.....	19
5.1.2	View Model	19
5.1.3	Controller.....	19
5.1.4	Service	20
5.1.5	Repository.....	20
5.1.6	Database	20
5.2	Application Process Sequence Diagram.....	21
6	Database Design.....	22
6.1	Mission Statement.....	22
6.2	Mission Objectives.....	22
6.3	Logical Database Diagram	23
6.4	Data Dictionary	23
6.4.1	Traveler.....	23
6.4.2	Message.....	24
6.4.3	TravelerRelationship	24
6.5	Foreign key integrity rules.....	25
6.5.1	Traveler.....	25
6.5.2	Message.....	25
6.5.3	TravelerRelationship	25
7	Test Plan.....	26
7.1	Unit Test.....	26
7.2	Integration Test	26
7.3	System Test	26
	References	27
Appendix 1	System Testing Forms	28

Abbreviations

GPS	Global Positioning System
MVC	Model View Controller
WCF	Windows Communication Foundation
xHTML	Extensible Hyper Text Markup Language
ORM	Object Relational Mapping
SOA	Service Oriented Architecture
DBMS	Database Management System
SQL	Structured Query Language
OS	Operating System
API	Application Programming Interface
IP	Internet Protocol
CSS	Cascading Style Sheet
CMS	Content Management System
REST	Representational State Transfer
CRUD	Create Read Update Delete
LINQ	Language Integrated Query
SOAP	Simple Object Access Protocol
IDE	Integrated Development Environment

1 Introduction

The purpose of this documentation is to describe the system design in a technical manner which includes a detailed description of its infrastructure and architecture. This document is based on the system's software requirements documentation and intended to be the foundation and guidelines of the system's implementation.

2 System Specifications

2.1 Technical Environment

The system will operate over Web environment with IIS 7.5 Web Server and Microsoft SQL Server 2008 R2 DBMS over Windows Server 2008 R2 OS on the server side, and with a known web browser on the client side (such as: FireFox, Internet Explorer).

The system will be mainly developed in C# .NET technology, JavaScript, xHTML and CSS.

2.2 System Integration

- The system will identify the traveler's location based on his IP address using an external GEO IP database by MaxMind.

2.3 Tools

The tools that will be used to develop the system are:

- Visual Studio 2010 IDE – C# .NET development tool
- GIT – Version control program
- FireBug – plug in to FireFox for examining and debugging HTML markup, CSS and JavaScript
- Selenium – Test automation tool for web application

2.4 Frameworks

The following frameworks will be used in the system development:

- ASP .NET MVC 3 – web application development framework
- WCF 4 – web services development framework
- Entity Framework – ORM framework
- Ninject - Inversion of Control container framework
- Log4Net – Logging framework
- LINQ – adds native data querying capabilities to .NET framework
- MSTest – Unit Testing framework
- Moq – mocking library that takes the advantage of LINQ for Unit Testing
- AutoMapper – automates object mapping between identical classes
- Validation Application Block 5.0 – validation framework integrated with WCF

3 Architecture Design

3.1 Multi-tier Architecture Model

The system will be developed in the three-tiered architecture partitioning, providing separation of concerns for easy maintainability, upgradability and platform dependency.

Tier	Layer	Part	Description
Client	Presentation	Views, Controls	User interface representation
		Resources	Images, Sound files, etc.
Web Server	Application	Controller classes	Control application flow
		View Model classes	Contain view data, view logic
		Repository classes	Facilitate simple access to database tables operations
		ORM classes	Represent database entity classes
		Service classes	Business logic and system features classes
Database Server	Data Storage	DBMS	Relational database
Web Services		Web Service API	Enables accessing web service via REST architecture

3.2 Layers Description

3.2.1 Presentation layer

The presentation layer will consist of dynamic content which is sent by the web server to the user's internet browser and then rendered by the client's browser. It's in charge of how the data is displayed, sorting and arranging it according to the user's preferences. This layer will be using the MVC architecture based on ASP.NET MVC 3 Framework. Each use case will have its own view.

3.2.2 Application layer

The application logic layer will provide the core functionality of the system according to the use case analysis and will be based on the Service Oriented Architecture using WCF 4 Framework to keep the system platform independent and enable easy integration.

3.2.3 Data Access layer

The data access layer will behave as a bus for data retrieval and persistence. Each function in this layer will consist of bare data retrieval and storage, without filtering or sorting.

3.3 Design Patterns

The system will make wide use of programming design patterns to enable efficiency, maintainability and interoperability. The main patterns that will be used are described shortly below:

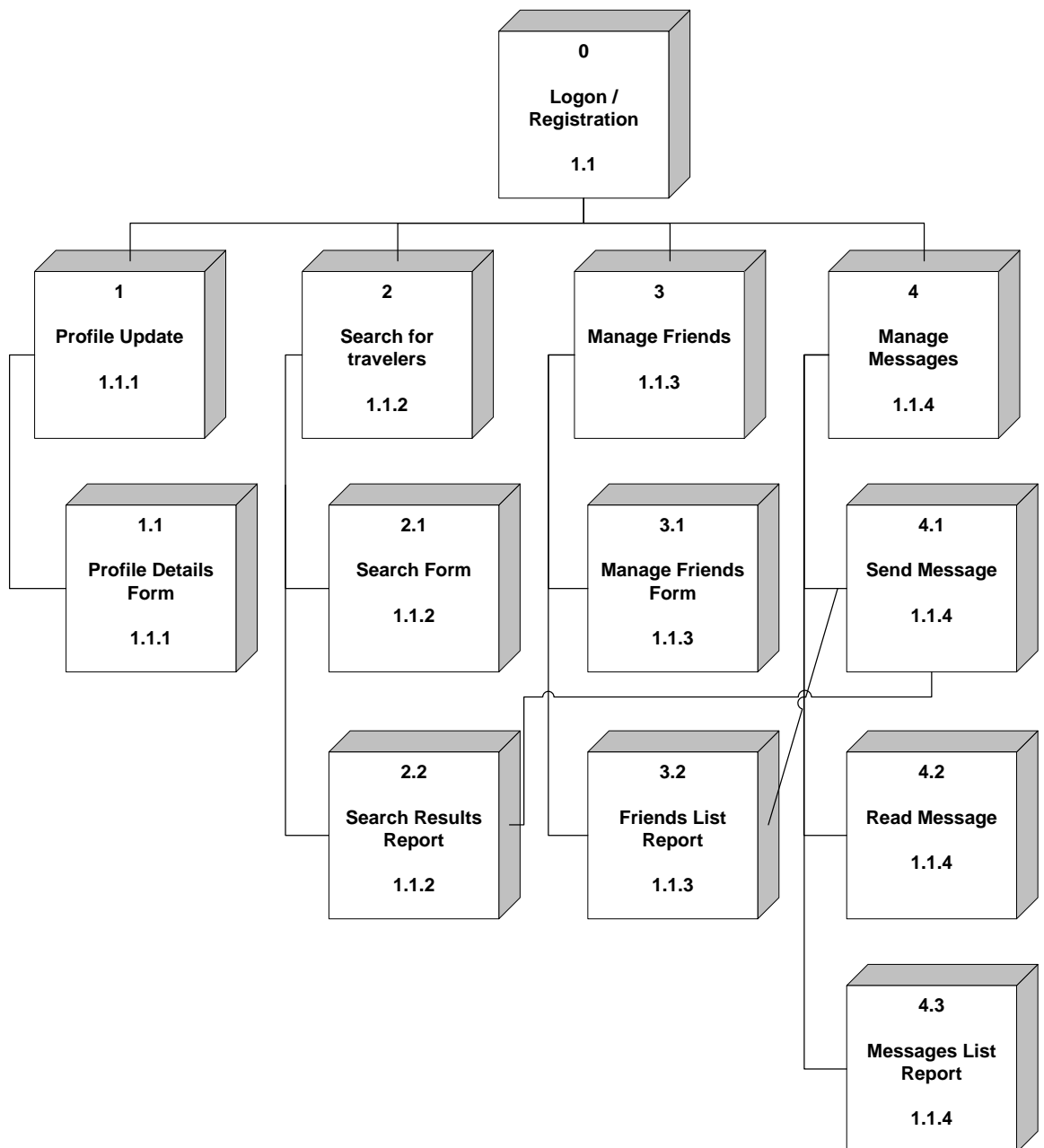
- **Dependency Injection** – helps decoupling software units through dependency of abstractions rather than concrete implementations.
- **Repository** – acts as in-memory collection completely isolating business entities from the underlying data infrastructure
- **Factory** – handling creation of objects without specifying the exact class

- **Domain Model** – object-oriented approach involving creation of an abstract model of real business domain, ignoring the underlying persistence infrastructure
- **Request Response** – design pattern for service oriented application dictating that each operation request and response should be defined by a class which is known by the client and server.

4 User Interface Design

The following diagram shows the navigation among each of the system's component the user will take.

4.1 Interface Structure Diagram



4.2 Interface Standards Design

4.2.1 Interface Metaphor

The system interface metaphor used is a cartoon drawing of the globe with the system's name around it. The globe represents the travelers around the world.



4.2.2 Interface Objects

The following names will be used as interface objects:

- **Traveler** – represents a traveler in the system
- **Message** – represents a message sent/received by a traveler
- **Profile picture** – represents a traveler's personal picture
- **Status message** – represents a message that will be displayed in search results next to travelers' profile pictures






4.2.3 Interface Actions

The following names will be used as interface actions:

- **Profile** – represents update profile function
- **Friends** – represents manage friends list function
- **Messages** – represents manage messages function
- **Search** – represents search for travelers function
- **Login** – represents logging in to the system function
- **Register** – represents registration of new user function
- **Send** – represents the confirmation to send a message
- **Save** – represents the confirmation to save changes to the database
- **Submit** – represents the submission of a form to the system

4.2.4 Interface Icons

The system will make use of common used icons which are already familiar to users from other web domains and applications. Such icons include:

-  Portrait icon to represent user's profile update action
-  Magnifying glass icon to represent search for travelers action
-  Group icon to represent the manage friends list action
-  Shutdown symbol icon to represent the log out action
-  Envelope icon to represent manage messages/send message action
- + Plus icon to represent add friend action

4.2.5 Input Design

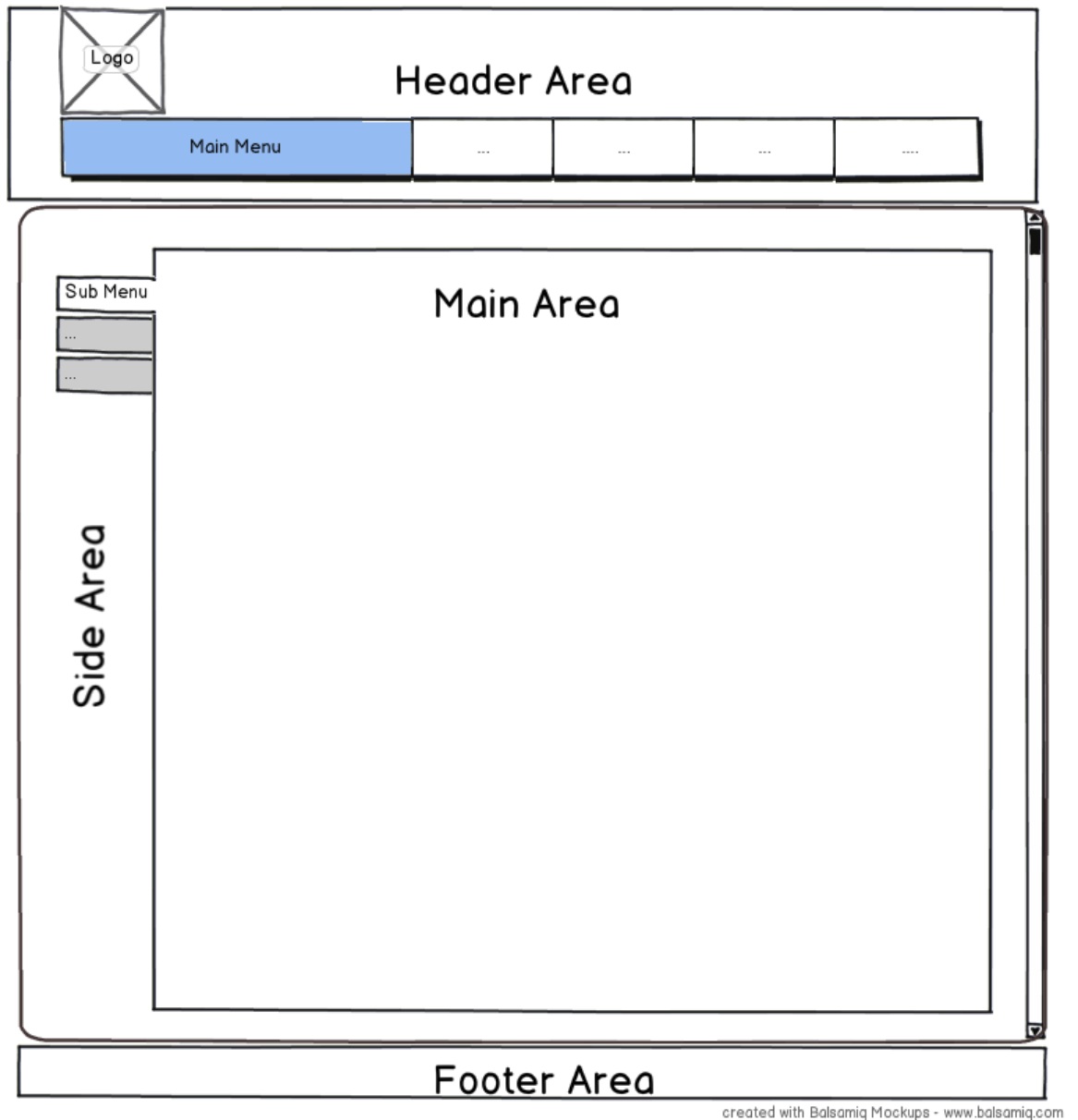
The system will use the built-in xHTML form controls for user input: textboxes, radio buttons, checkboxes, list boxes dropdown menus, etc. The system will validate user's input prior to processing it or storing it in the database and display appropriate messages to the user. Each required field will be marked with * and invalid input will be marked with a red underline per each invalid input field followed by a message.

4.2.6 Output Design

The system will use the built-in xHTML formatting elements to format system output to be rendered by the user's web browser.

4.2.7 Interface Template

The following interface template will be used throughout the system:



The header area will contain the main menu, logo and other possible navigation links. The Side area will contain sub menus of each feature. The main area will contain the part where the user will interact with the most, including forms, reports, etc. The footer will contain links and copyright information.

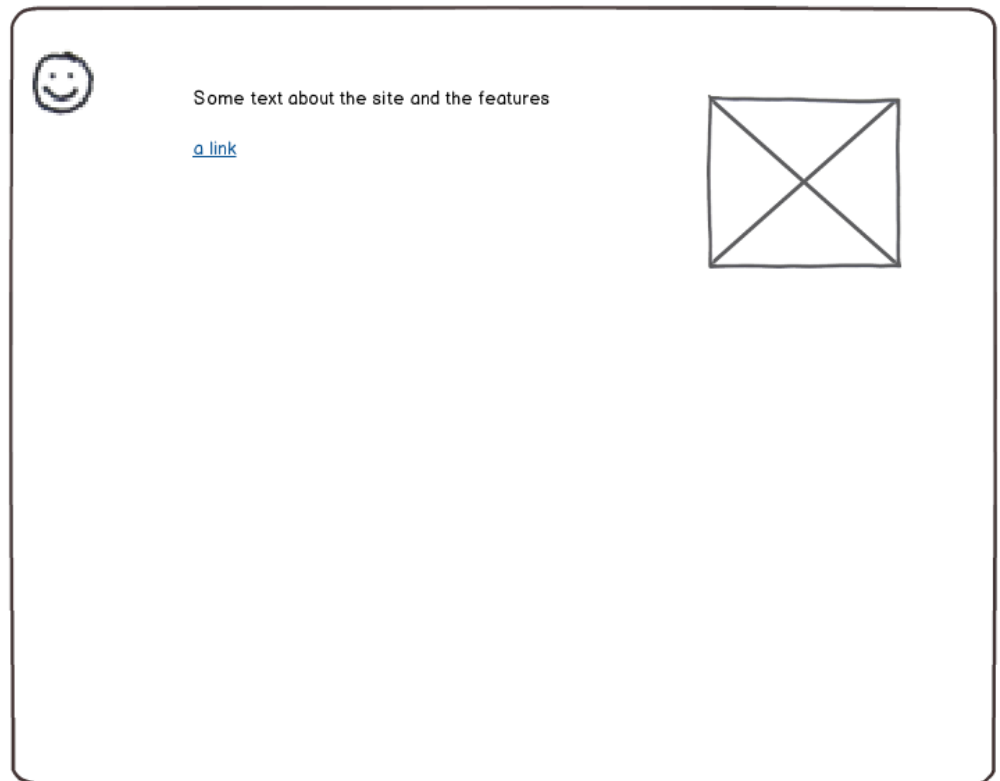
4.3 Interface Design Prototype

The following are user interface prototypes for each of the system's features.

4.3.1 Logon and landing page

TravelersAround

Email	<input type="text"/>	Register
Password	<input type="password"/>	<input type="button" value="Login"/>



created with Balsamiq Mockups - www.balsamiq.com

4.3.2 Registration

TravelersAround

Registration

First name

Last name

Birthdate

Gender

Email

Password


Retype password

created with Balsamiq Mockups - www.balsamiq.com

4.3.3 Update Profile

TravelersAround

Profile Search Friends Messages Log out



Profile picture

status: ▾
Available
Away

First name

Last name

Birthdate

Gender

Email

Password

Retype password

Unregister

created with Balsamiq Mockups - www.balsamiq.com

4.3.4 Search for travelers around

TravelersAround

Profile Search Friends Messages Log out

Available
 Everybody

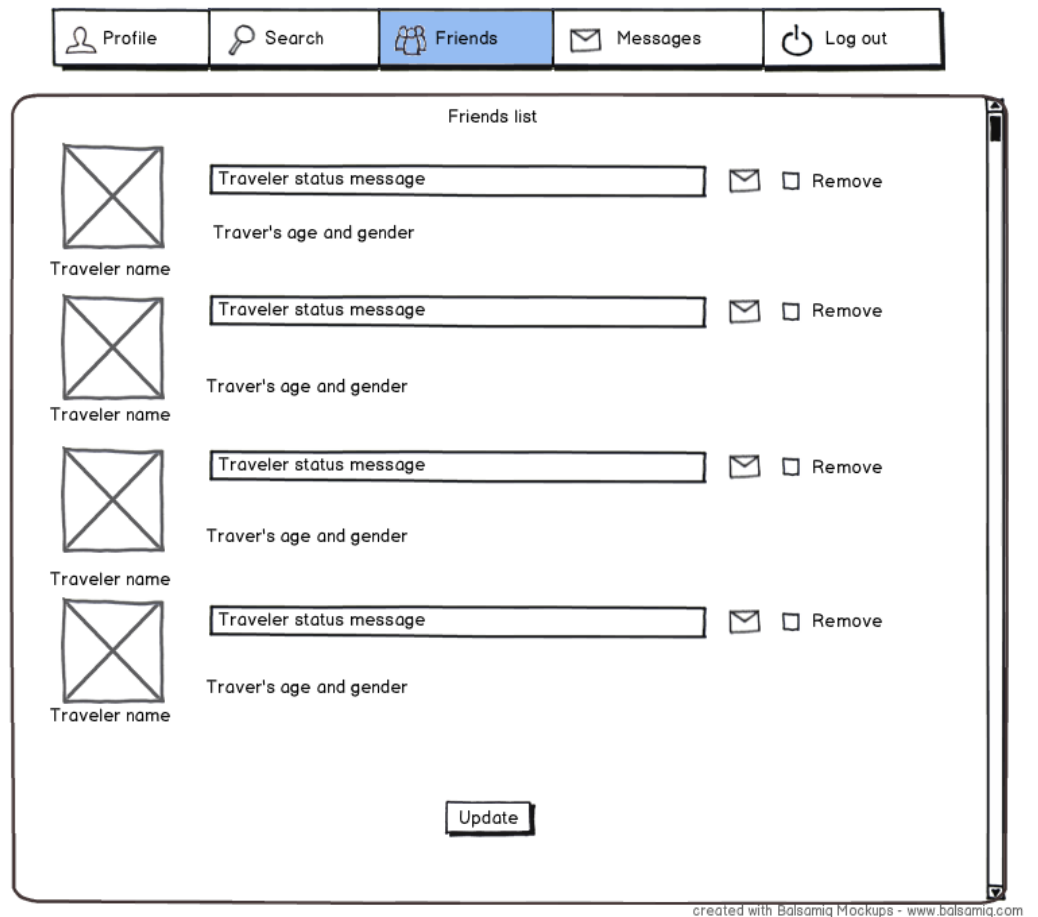
Found 4 travelers around you!

	<input type="text" value="Traveler status message"/>	+
Traveler name	Traver's age and gender	
	<input type="text" value="Traveler status message"/>	+
Traveler name	Traver's age and gender	
	<input type="text" value="Traveler status message"/>	+
Traveler name	Traver's age and gender	
	<input type="text" value="Traveler status message"/>	+
Traveler name	Traver's age and gender	

created with Balsamiq Mockups - www.balsamiq.com

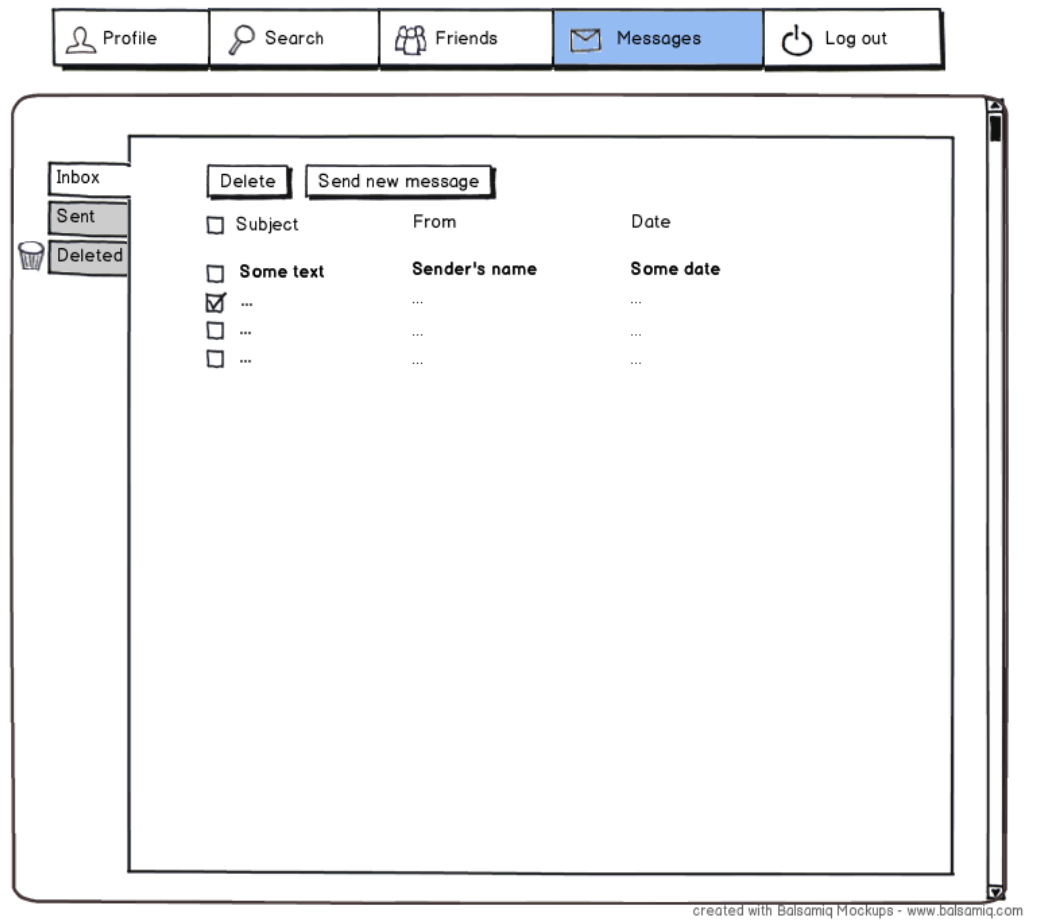
4.3.5 Manage friends list

TravelersAround



4.3.6 Manage messages

TravelersAround



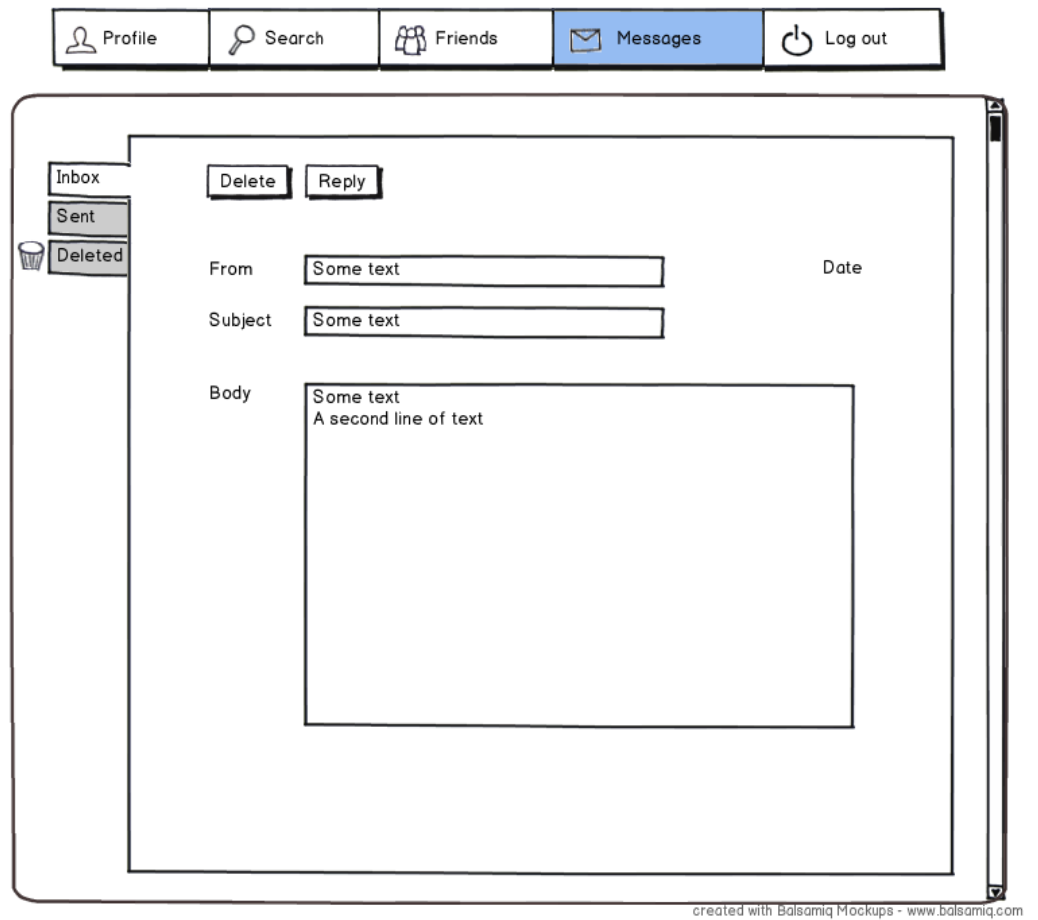
4.3.7 Send message

TravelersAround

The image shows a wireframe for a web application interface. At the top, there is a navigation bar with five items: Profile (with a person icon), Search (with a magnifying glass icon), Friends (with a group of people icon), Messages (with an envelope icon and highlighted in blue), and Log out (with a power icon). Below the navigation bar is a large rounded rectangle representing the main content area. On the left side of this area is a sidebar with three folders: Inbox, Sent, and Deleted (with a trash can icon). The main content area contains a message form with the following elements: a 'To' label followed by a text input field containing 'Some text'; a 'Subject' label followed by a text input field containing 'Some text'; a 'Body' label followed by a larger text area containing 'Some text' and 'A second line of text'; and a 'Send' button at the bottom left of the form. A small footer at the bottom right of the wireframe reads 'created with Balsamiq Mockups - www.balsamiq.com'.

4.3.8 Read message

TravelersAround



5 Application Design

The application design will be consistent among the system's features. Each feature will go through the same workflow which is described in the following sequence diagram. The application logic is divided into different components. Each component is in charge of a step in a process and combined together to fulfill a user's request.

5.1 Components Description

5.1.1 View

View is the main presentation unit that forms the user interface and will consist of xHTML markup constructed by the web server and sent back to the user as response. Each view is an ASPX file containing xHTML markup and C# code. View will only contain presentation related logic, such as sorting and ordering and define which controls should be shown. The view makes use of a *View Model*, which is described below, to display dynamic data.

5.1.2 View Model

View Model is a C# program class which describes and holds the data presented in a specific *View* and its metadata descriptions and requirements. The View Model is instantiated and populated by a *Controller*, which is described below. Each View Model in the application design will be constructed specifically per each user interface class analyzed in the software requirements documentation.

5.1.3 Controller

Controller is a C# program class which is in charge of each action's workflow. It receives a request and processes it by the appropriate action method. It is in charge of instantiating the appropriate *View Model* and performing validation, the workflow of each action and eventually presenting the right view to the user. A controller makes calls functions in a service class, which is described below.

5.1.4 Service

Service is a C# program class which forms business logic - the core features that the system offers. Each use case analyzed in the software requirements documentation is presented here as a method. The service class will be designed as a web service and behave as another tier in the system. It might be that this service will run on another web server than the web application itself. It will allow access to it via REST architecture. The service class calls functions from a repository class, which is described below.

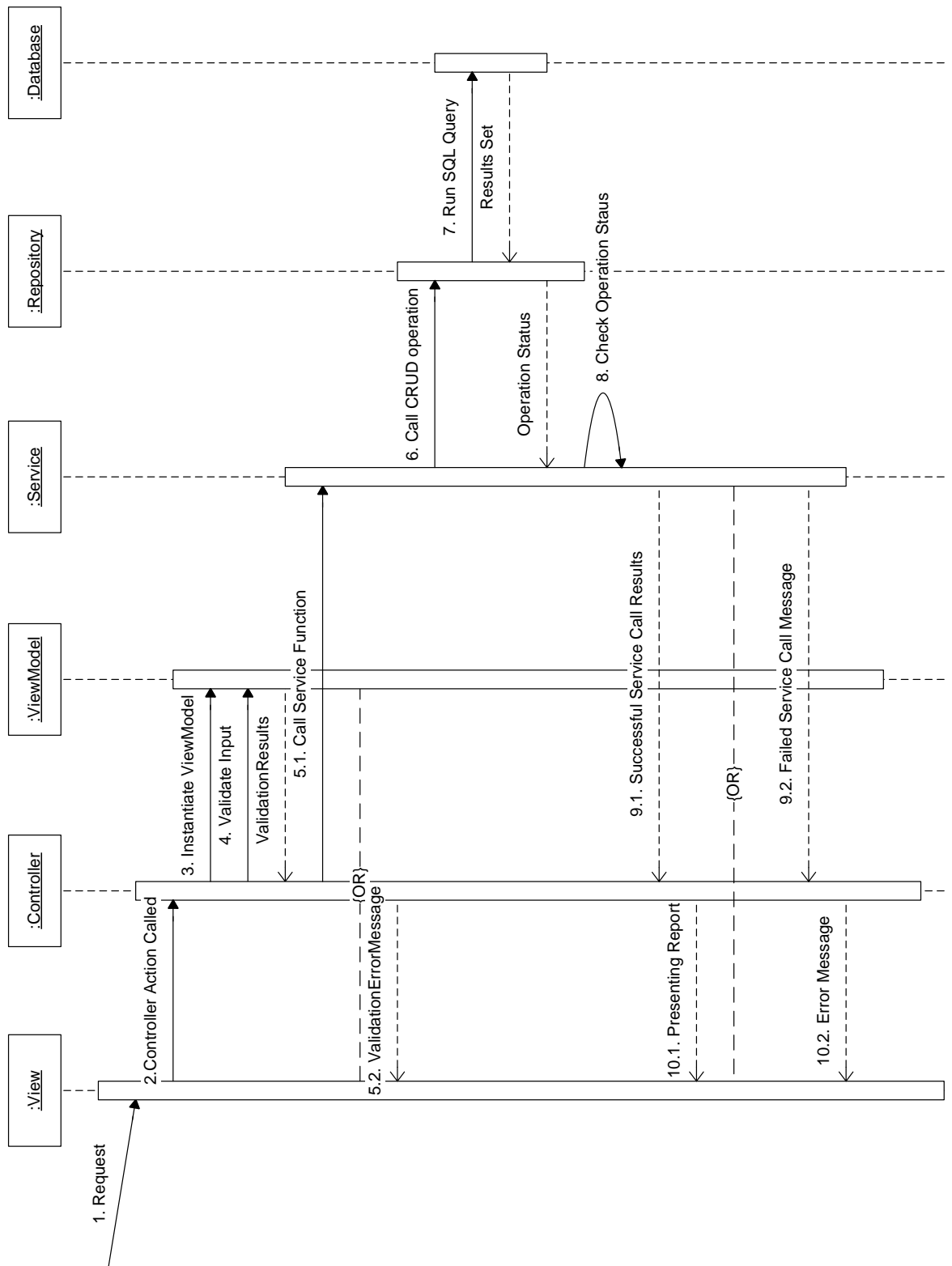
5.1.5 Repository

Repository is a C# program class which behaves as a data access layer to the database. The repository classes in this application design will use an ORM to database entities and tables. Repository classes will make CRUD calls to modify the database. Each call will be made using LINQ to SQL commands via the Entity Framework and generate dynamic SQL queries as needed.

5.1.6 Database

Database is the component where data will be persisted and maintained in the system. The application design will use Microsoft SQL Server 2008 R2. The database might reside on another server than the web server.

5.2 Application Process Sequence Diagram



The diagram shows the interaction among different components in the system and the order they are calling each other to form one process. Each feature in the system follows the above process sequence.

6 Database Design

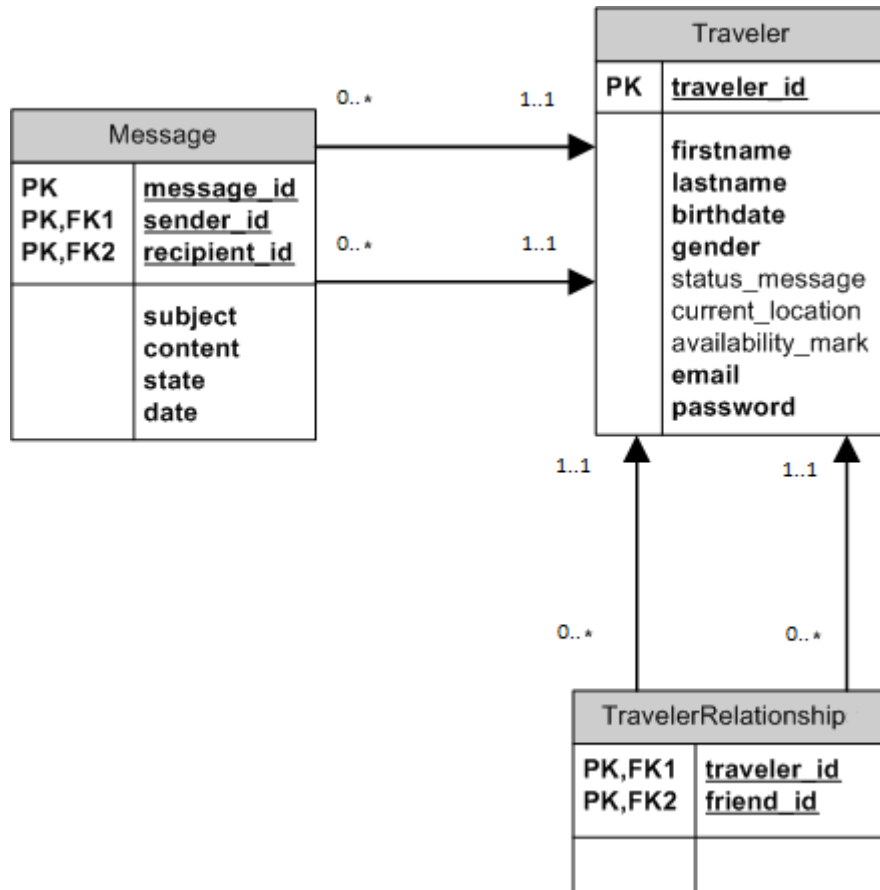
6.1 Mission Statement

The purpose of the database is to store information about travelers around the world and connects them together while they are on the go and physically in the same city, area, etc. The data will be kept up-to-date at all times to provide reliable information to travelers.

6.2 Mission Objectives

- Keep track of Traveler information
- Keep track of Message information
- Keep track of Traveler's relationships with other travelers

6.3 Logical Database Diagram



6.4 Data Dictionary

6.4.1 Traveler

Attribute	Type	Initial value	Allow null	Size	Description
Traveler_id	GUID	-	NO	-	PK. Unique identifier
Firstname	String	-	NO	50	Traveler's first name
Lastname	String	-	NO	50	Traveler's last name
Birthdate	Datetime	-	NO	-	Traveler's birthdate
Gender	Bit	-	NO	-	Traveler's gender. 0 defines male. 1 defines female.
Status_message	String	"I'm new here"	YES	250	Traveler's status message

Current_location	String	-	NO	-	Traveler's current location
Email	String	-	NO	50	Traveler's email address
Password	String	-	NO	8	Traveler's password

6.4.2 Message

Attribute	Type	Initial value	Allow null	Size	Description
Message_id	GUID	-	NO	-	PK. Unique identifier
Sender_id	GUID	-	NO	-	PK. FK1. Traveler ID of who sent the message
Recipient_id	GUID	-	NO	-	PK. FK2. Traveler ID of the recipient of the message
Subject	String	-	NO	150	Message's subject
Content	String	-	NO	500	Message's body text
State	Integer		NO	-	Message's state: read, unread
Date	datetime	-	NO	-	Date & time the message was sent

6.4.3 TravelerRelationship

Attribute	Type	Initial value	Allow null	Size	Description
Traveler_id	GUID	-	NO	-	PK. FK1. Traveler ID
Friend_id	GUID	-	NO	-	PK. FK2. Friend's Traveler ID

6.5 Foreign key integrity rules

6.5.1 Traveler

No foreign keys.

6.5.2 Message

- Foreign key *sender_id* referencing *traveler_id(Traveler)* on update cascade on delete no action
- Foreign key *recipient_id* referencing *traveler_id(Traveler)* on update cascade on delete no action

6.5.3 TravelerRelationship

- Foreign key *traveler_id* referencing *traveler_id(Traveler)* on update cascade on delete no action
- Foreign key *friend_id* referencing *traveler_id(Traveler)* on update cascade on delete no action

7 Test Plan

Testing is a crucial part to ensure the system's is working as it should and stands the required standards. This plan provides an overview of activities taken during the testing phase and their chronological order. Each test is assigned to a specific part in the system development process and describes who is in charge of it. The following are test types and they will be conducted in the order mentioned here.

7.1 Unit Test

Each unit or program class will be tested individually to make sure it has no errors and fulfils its purpose. This testing phase is done by “white box” testing, meaning that each test is done through examining the class code part and the way it functions. The tool that will be used in this phase is *Visual Studio IDE* along with *MSTest*, .NET Framework's built-in Unit Testing framework and *Moq*, a mocking framework, will be used to assist in this process. This test will be conducted by the developer.

7.2 Integration Test

In this phase, the developer will test the integration of each unit with another unit and examine its results. The same tool and frameworks will be used here to assist the test as in the Unit Testing phase with the addition of *Selenium*, a web browser automation tool. The test will be carried out right after Unit Testing phase. Some units, mostly third party units, will be treated as “black box”, meaning that the test is done without seeing the underlying program code. Any errors that may be found will be fixed right away and tested again.

7.3 System Test

This test is the last testing phase which basically consists of “black box” testing of the system as a whole, using the instructions provided in the test forms (See Appendix 1). It will take place at the project's closing meeting, the 30th of Sep. 2011 12:00 in Haaga-Helia, and be conducted by the steering group members, excluding the project manager, who's also the developer in this case. Any errors found in the outcomes of this test will be documented and fixed later on.

References

Dennis, A., Wixom, B.H. & Roth, R.M. 2006. System Analysis & Design. 3rd Edition. John Wiley & Sons, Inc.

Millett, S. 2010. Professional ASP.NET Design Patterns. Wiley Publishing, Inc., Indianapolis, Indiana.

Course Materials, Information System Development Project. SYS1TF080-7. 2011. Haaga-Helia. Helsinki.

Pukkila, V. 2009. Thesis: Hotel System with Java and MySQL. Haaga-Helia. Helsinki.

Mushimiyimana, S. 2009. Thesis: Tin Phong Sales Management System. Haaga-Helia. Helsinki.

Appendix 1 System Testing Forms

Registration

Step	Instructions	Expected Results	Errors/Exceptions	To be fixed?(Y/N)
1	Navigate to the main page by clicking the logo	Main page appears		
2	Click on register link	Registration form appears		
3	Fill up the form			
4	Click Submit button	The system grants access and navigates to Search page		

Logon

Step	Instructions	Expected Results	Errors/Exceptions	To be fixed?(Y/N)
1	Navigate to the main page by clicking the logo	Main page appears		
2	Fill up the form at the top right side	Registration form appears		
3	Click login button	The system grants access and navigates to Search page		

Search

Step	Instructions	Expected Results	Errors/Exceptions	To be fixed?(Y/N)
1	Navigate to the Search page by clicking it on the main menu	Search page appears		
2	Place a mark in the checkbox			
3	Click search button	The system gives a report of who's around the logged on traveler		
4	Click the plus icon next to one of the travelers to add him to your friends list	The system will notify that the traveler was added to friends list		

Update profile

Step	Instructions	Expected Results	Errors/Exceptions	To be fixed?(Y/N)
1	Navigate to the Profile page by clicking it on the main menu	Profile page appears		
2	Change the first name to John			
3	Click Save button	The system notifies that the save was successful		

Manage friends list

Step	Instructions	Expected Results	Errors/Exceptions	To be fixed?(Y/N)
1	Navigate to the Friends page by clicking it on the main menu	Friends page appears		
2	Place a mark in the checkbox of one friend			
3	Click Save button	The system notifies that the list was updated and shows the refreshed list		

Manage messages

Step	Instructions	Expected Results	Errors/Exceptions	To be fixed?(Y/N)
1	Navigate to the Message page by clicking it on the main menu	Message page appears		
2	Send a message by clicking on the send button	Send message page appears		
3	Fill up the form and choose a recipient from your friends list			
4	Click the send button to send the button	The system will notify that the message sent		
5	Go back to step 1			
6	Click on sent messages tab on the left hand side	Sent messages page appears		

	of the interface	showing the messages that just had been sent		
7	Click on the message's subject to open its content	Message's content message will appear		
8	Click the delete button	The message will be deleted from the list and the system will notify that the message was deleted and refresh the messages list		



HAAGA-HELIA
University of Applied Sciences

Implementation Documentation

TravelersAround

Version	1.0	
Created by	Nimrod Dayan	13.9.2011
Reviewed by	Juhani Välimäki	14.9.2011
Approved by	Juhani Välimäki	30.9.2011



Version	Date	Description	Author
0.1	13.9.2011	Added introduction and implementation content descriptions	Nimrod Dayan
0.2	20.9.2011	Added system workflow diagram and core units descriptions	Nimrod Dayan
0.3	25.9.201	Added database schema diagram	Nimrod Dayan

Table of contents

Abbreviations.....	1
1 Introduction.....	2
1.1 Implementation design.....	2
2 Implementation Content.....	3
2.1 Application Logic Layer.....	3
2.2 Data Access Layer.....	4
2.3 Service Layer.....	4
2.4 Presentation Layer.....	4
2.5 Unit Testing.....	5
3 Core Units.....	6
3.1 MembershipService.....	6
3.2 TravelersAroundService.....	7
3.3 HttpRequestAdapter.....	9
3.4 EFRepository.....	9
References.....	10
Appendix 1 : Database schema diagram	

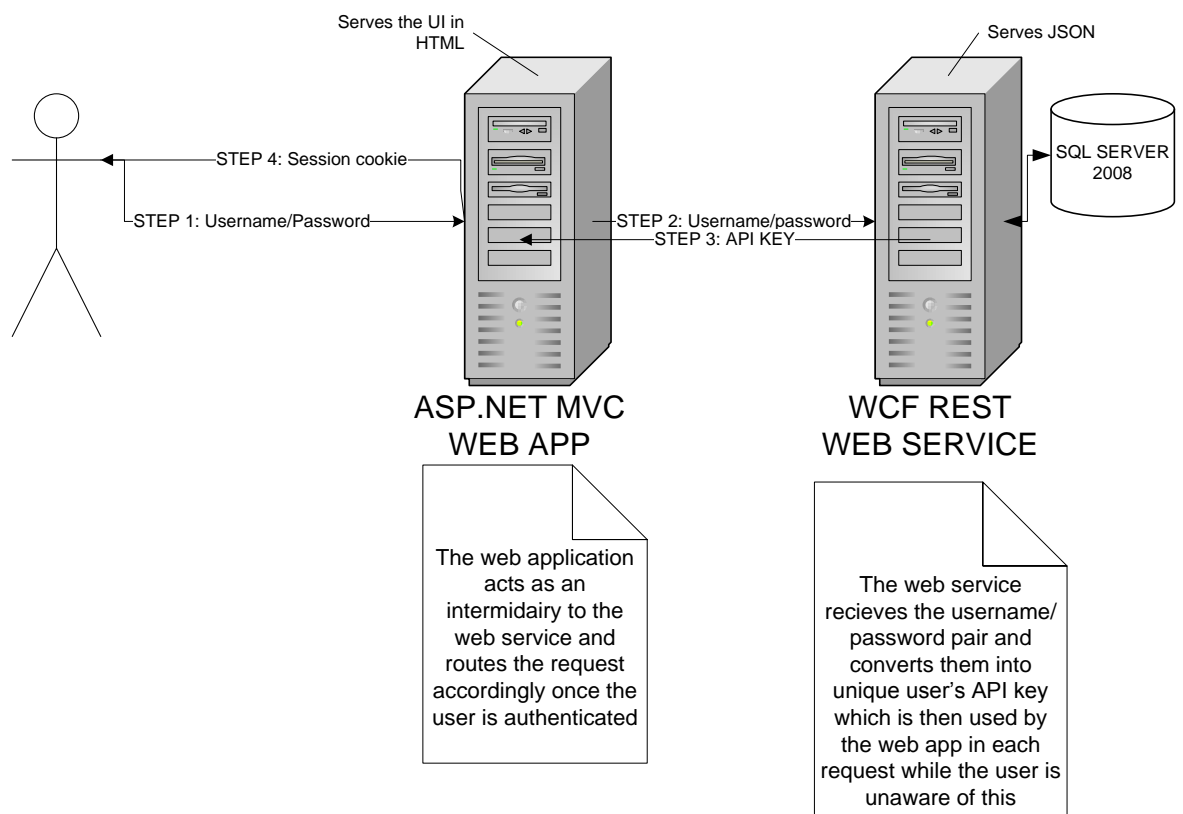
Abbreviations

GPS	Global Positioning System
MVC	Model View Controller
WCF	Windows Communication Foundation
xHTML	Extensible Hyper Text Markup Language
ORM	Object Relational Mapping
SOA	Service Oriented Architecture
DBMS	Database Management System
SQL	Structured Query Language
OS	Operating System
API	Application Programming Interface
IP	Internet Protocol
CSS	Cascading Style Sheet
CMS	Content Management System
REST	Representational State Transfer
CRUD	Create Read Update Delete
LINQ	Language Integrated Query
SOAP	Simple Object Access Protocol
IDE	Integrated Development Environment

1 Introduction

This document is a light version of the implementation phase of TravelersAround thesis project. It describes the implementation in general and explains how it works. It does not cover every unit of the system, but only the core units which are essential to know. The purpose of this documentation is to provide information on the implementation components and its structure for other developers or future reference for maintenance purposes.

1.1 Implementation design



The diagram shows the flow of data in the system. The user is unaware that the web application is communicating with the web service to process the requests. The web application role is purely to serve the UI, while not knowing how the requests are processed and what business rules and logic are applied. This design enables very light UI implementations and scalability for further UI developments such as mobile phones, etc.

2 Implementation Content

This chapter explains the folders structure of the implementation and the role of each assembly in the solution.

The source code is available to download from GitHub:

<https://github.com/Nimrodda/TravelersAround>

Lib – this folder contains external 3rd party reference libraries.

Src – this folder contains the source code of the system divided into assemblies.

Docs – this folder contains the source code documentation.

Builds – this folder contains release builds of the system.

As mentioned earlier, the implementation source code found in SRC folder is divided into separate assemblies a.k.a. Visual Studio projects. The implementation is divided into three logical layers. The following is the description of each layer and its assembly's roles:

2.1 Application Logic Layer

This layer contains the application business logic which follows Domain Driven Design principles.

TravelersAround.Infrastructure - contains infrastructure related classes which are generic for every system usage.

TravelersAround.Model – contains the entities, domain services, interfaces and business logic which are persistence ignorant.

TravelersAround.Resources – contains resources such, as localized text, that are used in the system.

2.2 Data Access Layer

This layer contains the data access strategy used in the system.

TravelersAround.Repository - contains the data persistence strategy using Entity Framework and SQLite adapter class.

TravelersAround.GeoCoding – contains the concrete implementation of location determination using *MaxMind GeoCoding* library.

2.3 Service Layer

This layer forms the gateway from the presentation layer to the application logic and data access layers. It is represented as a WCF web service, which is also divided into separate assemblies to provide scalability and maintainability.

TravelersAround.Contracts – contains the service contracts interfaces and their behaviors.

TravelersAround.DataContracts – contains service request/response data types and their validation rules.

TravelersAround.Service – contains the concrete implementation of each use case of the system. This assembly communicates among different assemblies in the system and acts as an intermediary between the client and the server.

TravelersAround.HTTPHost – contains the service host strategy based on WCF REST web service, which acts as the concrete endpoint for the client.

2.4 Presentation Layer

TravelersAround.ServiceProxy – contains the proxy to the service and provides a façade to the UI.

TravelersAround.WebMVC – contains the UI implementation based on MVC 3 framework. This assembly uses the ServiceProxy to make calls to the system's service.

2.5 Unit Testing

TravelersAround.Test – contains unit tests of different units in the system.

3 Core Units

This part will describe the core classes in the system and show code snippets.

3.1 MembershipService

This class is the implementation of the IMembershipService interface contract which is responsible for user authentication in the system. When a user (can be a computer or human) wants to use the services of the system, it must first call the Register or Login operation endpoints. Once the user is authenticated, an API key will be assigned, which will then be used to call the system operations.

The following code snippet shows the Login operation implementation:

```
LoginResponse response = new LoginResponse();
try
{
    //Validating traveler's membership
    bool isMember = _membership.ValidateUser(loginReq.Email, loginReq.Password);
    if (isMember)
    {
        Guid travelerID = _membership.GetUserTravelerID(loginReq.Email);
        //Loading traveler's profile
        Traveler traveler = _repository.FindBy<Traveler>(t => t.TravelerID == travelerID);

        //Issuing an API key and storing in cache
        APIKeyService apiKeySvc = new APIKeyService(_repository, _apiKeyGen);
        string travelerApiKey = apiKeySvc.GetUniqueApiKey(loginReq.Password);
        apiKeySvc.Store(travelerApiKey, traveler.TravelerID);
        traveler.ApiKey = travelerApiKey;

        _repository.Save<Traveler>(traveler);
        _repository.Commit();

        response.NewMessagesCount = traveler.Messages.Count(m => m.IsRead == false && m.FolderID == (int)FolderType.Inbox);
        response.ApiKey = travelerApiKey;
        response.MarkSuccess();
    }
    else
    {
        response.ErrorMessage = R.String.ErrorMessages.InvalidCredentials;
    }
}
catch (Exception ex)
{
    ReportError(ex, response);
}
return response;
}
```

3.2 TravelersAroundService

This class is the implementation of the `ITravelersAroundService` interface contract which contains operations that map to the use cases defined in the software requirements analysis. The user is required to provide an API key per each call to this service; otherwise the user will receive an error message indicating that the user is unauthorized.

The following code snippet shows the Search operation implementation, which shows the algorithm used to determine the currently online travelers in the area of the user. The algorithm uses the *Haversine* formula to calculate distances between two points in a sphere (Earth in this case). The formula is called by this operation explicitly through a database stored procedure for fast response (from optimization reasons).

```
public SearchResponse Search(bool includeOfflineTravelers, int index, int count, string
ipAddress = null, double lat = 0, double lon = 0)
{
    SearchResponse response = new SearchResponse();
    LocationService locSvc = new LocationService(_locationDeterminator,
_repository, _geoCoder);
    APIKeyService apiKeySvc = new APIKeyService(_repository, _apiKeyGen);

    try
    {
        Traveler currentTraveler = _repository.FindBy<Traveler>(t =>
t.TravelerID == _currentTravelerId);
        if (!String.IsNullOrEmpty(ipAddress) || (lat != 0 && lon != 0))
        {
            //This part is optional. It will happen only if the client will ex-
            plicitly the location, either by IP address or coordinates
            //otherwise the IP address of the client who made the request will
            be used to determine the location

            GeoCoordinates coords = new GeoCoordinates { Latitude = lat, Longti-
tude = lon };

            if (!String.IsNullOrEmpty(ipAddress))
            {
                locSvc.UpdateTravelerCoordinates(currentTraveler, ipAddress);
            }
            else if (currentTraveler.IsLocationChanged(coords))
            {
                currentTraveler.Latitude = coords.Latitude;
                currentTraveler.Longitude = coords.Longitude;
                currentTraveler.City = coords.City;
                currentTraveler.Country = coords.Country;
            }
        }
        else
        {
            //Normally this will happen
            locSvc.UpdateTravelerCoordinates(currentTraveler, APIKeySer-
vice.CurrentTravelerIPAddress);
        }
        //Updating traveler's location
    }
}
```

```

        _repository.Save<Traveler>(currentTraveler);
        _repository.Commit();

        //All online travelers in the system
        IEnumerable<Guid> allOnlineTravelersFromCache = api-
KeySvc.GetCurrentlyActiveTravelers();

        //Raw results of travelers around - including offline users
        PagedList<Traveler> travelersAroundRawResults =
locSvc.GetListOfTravelersWithin(RADIUS, index, count, currentTraveler);

        if (!includeOfflineTravelers)
        {
            //Gets only travelers around who are online at this moment out of
the 10 rows from the DB
            var onlineTravelersAround = travelersAroundRawRe-
sults.Entities.Where(t => allOnlineTravelersFromCache.Contains(t.TravelerID));

            //if we have another page from the DB (10 more travelers) and the
maximum rows for the page hasn't been reached then loop until
//all the rows for the page are full or until the last page has
reached
            while (travelersAroundRawResults.HasNext && onlineTravelersA-
round.Count() < count)
            {
                index = index + count; //index for the next page
                //get next page from the DB
                travelersAroundRawResults =
locSvc.GetListOfTravelersWithin(RADIUS, index, count, currentTraveler);
                //take only as many as needed to fill up the page
                var onlineTravelersFromNextPage = travelersAroundRawRe-
sults.Entities.Where(t => allOnlineTravelersFromCache.Contains(t.TravelerID)).Take(count
- onlineTravelersAround.Count());
                //pile up the results
                onlineTravelersAround = onlineTravelersA-
round.Concat(onlineTravelersFromNextPage);
            }
            travelersAroundRawResults = onlineTravelersAround.ToPagedList(index,
count);
        }

        //Marks which travelers are online
        for (int i = 0; i < travelersAroundRawResults.Entities.Count; i++)
        {
            if (allOnlineTravelersFrom-
Cache.Contains(travelersAroundRawResults.Entities[i].TravelerID))
            {
                travelersAroundRawResults.Entities[i].IsOnline = true;
            }
        }

        //Converts to the suitable data contract for serialization
        response.Travelers = travelersAroundRawRe-
sults.ConvertToTravelerViewList();
        response.MarkSuccess();
    }
    catch (Exception ex)
    {
        ReportError(ex, response);
    }
    return response;
}

```

3.3 HttpRequestAdapter

This class is a utility class used by the ServiceProxy assembly to make HTTP requests. As the Web Service uses REST architecture over HTTP, The ServiceProxy has to make calls to it via the HttpRequestAdapter class. Therefore, this class' role in the system is crucial. The class take care of serialization of primitive and complex C# data types and then sends them according to the required HTTP method.

The following is code snippet of the generic method WebHttpPostRequest which receives a C# object of any kind, serializes it and then sends it to the defined Uri. The results will be serialized to the specified generic type.

```
public static ResponseType WebHttpPostRequest<ResponseType>(string baseUrl, string uri, object requestObject, string queryString = "")
{
    HttpWebRequest invokeRequest = WebRequest.Create(String.Concat(baseUrl, "/", uri, queryString)) as HttpWebRequest;

    invokeRequest.Method = "POST";
    invokeRequest.ContentType = "application/json";

    byte[] requestBodyBytes = SerializeToJSON(requestObject);
    invokeRequest.ContentLength = requestBodyBytes.Length;
    using (Stream postStream = invokeRequest.GetRequestStream())
        postStream.Write(requestBodyBytes, 0, requestBodyBytes.Length);

    HttpWebResponse response = invokeRequest.GetResponse() as HttpWebResponse;
    return DeserializeFromJSON<ResponseType>(response.GetResponseStream());
}
```

3.4 EFRepository

This class is the implementation of IRepository interface, which is in charge of data persistence in the system. The repository takes any entity type which is of type IAggregateRoot and performs database operations accordingly.

The following code snippet shows the implementation of the generic method FindBy which takes in a lambda expression predicate and the required returned type as an input:

```
public TEntity FindBy<TEntity>(Expression<Func<TEntity, bool>> predicate) where TEntity : class
{
    if (predicate != null)
    {
        return _dataContext.CreateObjectSet<TEntity>().FirstOrDefault(predicate);
    }
    else
    {
        throw new InvalidPredicateException();
    }
}
```

References

URL: <http://www.movable-type.co.uk/scripts/latlong.html>

“Haversine formula” 10/9/2011

URL: <http://blogs.msdn.com/b/endpoint/archive/2010/01/07/getting-started-with-wcf-webhttp-services-in-net-4.aspx>

“Getting Started with WCF WebHttp Services in .NET 4 - The .NET Endpoint”
13/9/2011

URL: <http://msdn.microsoft.com/en-us/library/bb412179.aspx> “How to serialize JSON” data 15/9/2011

Millett, S. 2010. Professional ASP.NET Design Patterns.
Wiley Publishing, Inc., Indianapolis, Indiana.

Appendix 1 : Database schema diagram

