**SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE**

**CSC3002 – COMPUTER SCIENCE PROJECT**

**Dissertation Cover Sheet**

A signed and completed cover sheet must accompany the submission of the Software Engineering dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Student Name: **Michal Wasinski**  Student Number: **40043301**

Project Title: **Car Sharing App** (Specified by Kainos)

Supervisor: **Dr Peter Kilpatrick**

**Declaration of Academic Integrity**

Before signing the declaration below please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

**I declare that I have read both the University and the School of Electronics, Electrical Engineering and Computer Science guidelines on plagiarism - http://www.qub.ac.uk/schools/eeecs/Education/StudentStudyInformation/Plagiarism/ - and that the attached submission is my own original work. No part of it has been submitted for any other assignment and I have acknowledged in my notes and bibliography all written and electronic sources used.**

*Student’s signature* *Date of submission*

Contents

[1. Introduction. 3](#_Toc385231962)

[1.1 Problem description. 3](#_Toc385231963)

[1.2 Problems with current approach 3](#_Toc385231964)

[1.3 Solution 4](#_Toc385231965)

[1.4 Advantages of the new approach 5](#_Toc385231966)

[1.5 Goals 5](#_Toc385231967)

[1.6 Requirements 5](#_Toc385231968)

[Must haves: 6](#_Toc385231969)

[Should haves: 6](#_Toc385231970)

[Could haves: 6](#_Toc385231971)

[Would like to haves: 6](#_Toc385231972)

[1.7 Requirement Modifications. 7](#_Toc385231973)

[1.8 Hardware & Software Platform. 8](#_Toc385231974)

[1.9 Development Languages & Technologies used. 8](#_Toc385231975)

[2. Specification 9](#_Toc385231976)

[2.1 Data Model 9](#_Toc385231977)

[2.1.1Data Model Architecture 9](#_Toc385231978)

[2.1.2 Data Models 9](#_Toc385231979)

[2.2 Function Definitions 16](#_Toc385231980)

[3. Design 19](#_Toc385231981)

[3.1 User Interface Design 19](#_Toc385231982)

[3.1.1 Mobile Device Considerations. 19](#_Toc385231983)

[3.1.2 Android Application. 20](#_Toc385231984)

[3.2 System Software Design. 25](#_Toc385231985)

[3.2.1 Top-Level Components. 25](#_Toc385231986)

[3.2.2 Multi-tier architecture. 26](#_Toc385231987)

[3.2.3 Data Access Layer 27](#_Toc385231988)

[3.2.4 WCF Web Service 30](#_Toc385231989)

[3.2.5 Security 31](#_Toc385231990)

[3.2.6 Google Cloud Messaging (GCM) 31](#_Toc385231991)

[3.2.7 Google Maps 32](#_Toc385231992)

*Acknowledgments*

Special thanks to Seamus Sans and Garth McFarland from Kainos Belfast, who provided assistance as project mentors, and Dr Peter Kilpatrick from Queen’s University Belfast, who assisted as the project supervisor.

*Abstract*

This dissertation documents the software development process for a project specified by the local software company ‘Kainos’ from initial concept to specification, design, implementation and to conclusion. The aim of the project was to produce a mobile application that allows its users to share their cars.

The implementation produced satisfies all the core functionality and majority of the additional features specified in the requirements section. This resulted in a system that satisfied the requirements of both project mentors and the project supervisor.

# Introduction.

The aim of this project is to create a car sharing mobile application. Using their handheld devices, the applications’ users will be able to advertise their journeys and request a seat in ones they wish to join.

The concept of car sharing has received a lot of attention over the last few years. The idea is heavily supported by the Intelligent Energy Europe commission as it promotes efficient use of existing resources, contributes to the reduction of CO2 emissions and also helps to reduce the number of cars on the road by utilizing existing car space more efficiently.

## Problem description.

The following project brief, provided by Kainos, a local software development company in Belfast has been used to derive my problem description:

**Project aim:  *Create an app to facilitate the matching up of car sharing participants.***

***The application should allow users to advertise or search for car sharing participants. It should allow users to enter start/end locations, route, travel times.  Once a match is made the driver should be notified so they can approve. Once approval is received the application should provide a way for the driver and passenger to communicate.***

***The application could be extended by gamifying the driver role. The application could reward points to drivers for lift sharing and display a leader board of the results.***

For the original problem description, please refer to the appendices. //TODO – insert appendix reference.

## 1.2 Problems with current approach

Despite a number of well-established car sharing services already in existence, users suffer from a lack of a truly mobile platform bringing together the features of a car sharing service and combining them with the advantages that mobile devices offer. The most notable pitfalls include:

- Users must perform a manual search for every journey they wish to join for thus wasting their time for a feature that could be automated.

- Users are limited to a range of cities and locations recognised by the system often unable to search for journeys whose departure and destination points are not near their desired locations.

- Automatic push notifications should be implemented along with standard email notifications to alert the user of a relevant event immediately rather than requiring them to read their inbox.

- Users might not feel safe while travelling in a car with stranger whom they never met before.

In order to measure the usability of the application, a benchmark against other car-sharing services has been established. The below list represents a common set of features offered by all other car-sharing services. It’s important to note however, that the below list represents an absolute core set of features and a successful car sharing-service should implement additional functionality in order to increase its usability and appeal towards the user.

- **Login/Registration.** Allowing users to create a new account and log in.

- **Advertising user’s own journeys.** Providing the functionality to specify departure date, time and number of available seats.

- **Searching for journeys.** Being able to specify the departure and destination points for the desired journey as well as departure date and time.

- **Contacting drivers.** Being able to contact the journey’s driver and request a seat in the car.

- **Responding to requests.** Drivers must be able to view profiles of users who sent the request and make a decision either accepting or denying it.

## 1.3 Solution

A truly mobile car sharing application will utilise the features provided by the Android framework to create a user friendly, secure and fully functional car sharing platform. Using their Android handheld devices, users would be able to search for and offer journeys that will immediately become visible to other users of the system. The app aims at bringing together everyone who wishes to participate in car sharing through various in-app social features that will allow users to manage their friends, carry out real-time conversations as well as rate drivers and view feedback left by others.

To provide for better journey matching functionality, the application will include a search engine which will be based on a mathematical formula that will be able to perform intelligent location-aware search rather than string comparison and will work independently of the address format entered by the user. In addition, the search function will be extended to make use of Google Maps API in order to provide a location aware system that’s able to provide users with recommendations when results matching their exact criteria are currently unavailable. In addition, users will not be constrained to a specific address format which can be advantageous in a situation where only parts of the desired address is known. The geocoding system used inside the app will be flexible enough to translate even a partial or incomplete address into a real-world location.

Personalisation features will give users the ability to create their own journey preferences and be automatically notified by the system when a journey matching their preferences becomes available.

Through numerous social features, users are able to maintain a list of their friends, carry conversations in real time using the app’s built in instant messenger as well as view profiles of other users and be able to change their own privacy settings essentially controlling what aspects of their profile are visible to their friends and other users of the system.

To improve the sense of security, the app features a rating system where users can leave feedback and rating for the drivers. Global leader-board of the best drivers together with their feedback and scores will be maintained by the system and visible to all users.

## 1.4 Advantages of the new approach

This application will benefit the end user in the following ways:

* The application’s search engine, being one of the most important components of the system, will need to be able to provide more than just results that are exact match for user’s search criteria. It will need to be able to perform intelligent-search that will be able to provide journey recommendations based on user’s location. To achieve this, the search engine will be implemented as a mathematical function that will perform distance analysis. In addition, the search engine will work independent of the address format entered by the user as it will only operate on latitude and longitude values. For the user, this means that the application will automatically consider journeys departing from places nearby thus providing journey recommendations.
* The concept of journey templates will free the user from the requirement of performing explicit search when wish to find a journey. Journey Templates will act as subscription events where the user will be able to specify that they wish to be automatically notified by the system when a journey posted by another user matching their criteria has been advertised. This will not only save a lot of user’s time but also reduce the unnecessary server load and preserve the battery life of users’ Android device.
* The user’s sense of security will be improved through various social features built-in to the application. The ability to leave and read driver’s feedback left by other users together with a score rating will provide them with an insight into the driver’s past performance history. Creating private journeys only visible to user’s friends will ensure only known individuals whom the user trusts will be able to apply.
* Improved user interaction will be achieved with the help of instant-messenger and chat rooms built into the application. Users will be able to exchange messages in real-time with their friends and all of the passengers participating in a journey will be able to carry out a conversation in a multi-user journey chat room even if they are not in one another’s friends lists.

## 1.5 Goals

The core goal is to implement the entire solution consisting of the Android app, WCF web service as well as the administrator’s panel by 15th of May 2014.

A working prototype is scheduled to be presented on the week beginning 9th of December.

## 1.6 Requirements

The below list of requirements is a direct result of the requirements elicitation process which involved interviewing potential users of the app and was carried out before the development phase. The MoSCoW technique has been used as means of prioritisation for each individual piece of functionality.

It’s important to note that some of the requirements from the original requirements list have been modified as a result important design decisions made in the initial stages of the development process. Modified requirements represent requirements whose functionality has been altered to provide best possible functionality and usability to the user. Each of the modified requirements have been described in chapter 1.7.

### Must haves:

Represent requirements that must be satisfied in order for the final solution to be considered successful.

1. User registration **(Modified)**
2. User login **(Modified)**
3. Searching for journeys by start & end locations as well as date and be able to specify other search criteria.
4. Ability for users to post new car share listings and specify dates, locations and fee.
5. A web based admin panel to allow Administrator to log in and manage the system. **(Modified)**
6. Save a list of user trips. **(Modified)**
7. Exchange messages with other users via the application.
8. Implementation of Google Maps API to find cities and plot routes on the map.
9. Quick search facility with locations and dates based on user defined search criteria.**(Modified)**

### Should haves:

1. Represent high-priority items that should be included in the final solution.
2. Advanced search options such as: women only, smokers, type of vehicle, fee, and number of seats.
3. Rating system based on passengers experience with leader board.

### Could haves:

1. Represent a requirement which is considered desirable but not necessary.
2. Service in the background with notifications when a car share becomes available. **(Modified)**
3. Specify the main stops in the journey.
4. Allow users to specify a radius from the start and end locations in miles.
5. Allow users to specify a city region from the start and end locations in miles. **(Deleted)**
6. Instant messaging feature to allow app users exchange messages in real time.
7. Search for car share listings with the help of GPS to find the one with nearest start location. **(Modified)**

Would like to haves: Represents a requirement which is unlikely to be included in the final solution but represents functionality that could be implemented in the future release.

1. Live driver tracking using GPS.
2. Possible integration with Facebook and Google+

## 1.7 Requirement Modifications.

This section explains the changes that have been applies to the original requirements.

1. User Registration **(Modified)**

User registration has been separated from user login and moved into its own requirement. This is due to the considerable amount of work that is involved when registering a new user as it should be seen in chapter 4.

1. User Login **(Modified)**

The decision to move User Login into its separate requirement was influenced by the application’s session management feature as described in chapters 3 & 4. Session management will introduce additional an login function allowing the user to automatically log-in with their security token without the need to provide their username and password.

1. A web based admin panel to allow Administrator to log in and manage the system. **(Modified)**

After reviewing the concept of the applications administrator panel with both of my supervisors, Seamus Sands and Garth McFalrand of Kainos, it has been unanimously decided that the admin panel should be instead implemented as a desktop application. The admin panel should not be available to the outside world nor should it be visible to the regular end user. It is for these reasons that the administrator’s panel should be accessible via a third-party app.

1. Save a list of user trips**. (Modified)**

This requirement has been modified as a result of an important design decision made at an early stage of the development phase. For security purposes, all user information will be stored in a remote database that will be accessed through a web service residing in the service layer. Accessing information remotely will not only allow me to preserve the principles of multi-tier application design but will also allow the Android app to always stay in sync with the web service by downloading the most up-to-date data each time the user logs in. Another advantage is that user accounts will not in any way be tied to a specific Android device meaning users could log into the system from any Android device and access their information regardless.

1. Quick search facility with locations and dates based on user defined search criteria**.(Modified)**

The quick search facility has been replaced with user defined journey templates. Those templates are then used by the web service to notify the user once a journey which matches the template parameters has been offered. They can however, also be used to perform a one-click search simply by holding the finger on the desired template for a few seconds.

1. Service in the background with notifications when a car share (journey) becomes available. **(Modified)**

The concept of a service running in the background checking for new data has been replaced with push notifications provided by the Google Cloud Messaging (GCM) service. Sending push notifications directly to the phone instead of polling the server at regular intervals is a much more effective and efficient way of synchronising the application’s state and retrieving the latest information. It reduces the server load by making the Android application only call the web service when there is new data waiting to be downloaded. This also has a positive effect on the device’s battery life since web requests are computationally expensive.

1. Allow users to specify a city region from the start and end locations in miles. **(Deleted)**

With the help of Google Maps API and location aware search, there is no need to ask the user to specify the city region of their desired location. Instead, users will be asked to enter a specific location and provide a radius in miles which will not only extend the search area but also provide a higher likelihood of finding journey that matches user’s criteria.

1. Search for car share listings (journeys) with the help of GPS to find the one with nearest start location. **(Modified)**

When user performs a journey search or when they create a new journey template, they are required to specify the departure and destination points. The Android application will provide the user with special GPS buttons which will acquire user’s current location and perform geocoding to extract the address. This will save users from the effort of entering the address manually.

## 1.8 Hardware & Software Platform.

The entire solution will consist of four major components, the client application running on the mobile device, the web service which the mobile device will interact with, the database where all the information will be stored and finally the administrator panel giving the system administrator super user privileges. Below is a detailed breakdown of the hardware and software platforms on which each of the above components will be deployed.

* **Mobile application.** Developed using Android’s native environment deployable on any Android device. Choosing native Android environment over any of the “Write once deploy everywhere” solutions will provide full unrestricted access to all of the features offered by the Android framework without any performance penalty providing full access to the device and its hardware components.
* **Web Service.** Based on WCF REST 4.0, the web service will provide a scalable and extensible communication channel between the mobile application and the database where all the user and journey information will be stored. The API exposed by the service will make it possible to additional clients based on different platforms such as iOS to be developed.

The web service will utilise HTTP as its primary transport protocol making use of the HTTP verbs such as POST and GET.

The web service will be deployed to the cloud services provided by Amazon Web Services (AWS) which will provide for secure and scalable server infrastructure.

* **Database & Data Access Layer.** Deployed using Microsoft’s SQL Server with Entity Framework on top will provide for rapid database model modifications and regeneration when required. Entity Framework will provide a means of accessing the database data in terms of entities allowing to focus primarily on the business logic instead of writing SQL queries whilst providing protection against SQL injection attacks.
* **Administrator Panel.** A WPF MVC/MVVM Windows application deployable on any Windows machine or any Linux machine that can simulate Windows environment. For exclusive access by the administrator.

## 1.9 Development Languages & Technologies used.

The mobile application will be developed using Android’s native language Java using Google’s Android Studio IDE which provides full support for writing Android applications. The entire backend stack of the system, that is the web service and the data access layer will be developed using .NET 4.5 and C# as the development language.

# Specification

This section focuses on defining the data model which is derived from the problem specification described in chapter 1. It also provides specification of the main functions of the system and possible error conditions.

## Data Model

The data models used in this section will serve as foundation for developing the application’s domain objects. It’s important to note that these models are not the exact representation of the application’s domains and should be considered as a guideline.

### 2.1.1 Data Model Architecture

The data model will be logically separated into two distinct groups. The first group will contain a collection of Data Transfer Objects or (DTO’s), and the second group will consist of the domain objects which the database framework will manipulate on.

Data Transfer Objects will be used in web service calls as a means of transferring information from the client to the web service without exposing applications’ domain objects to the outside world. A properly designed DTO can reduce the number of web service calls that the client needs to make to perform a certain task by encapsulating more information in a single call to the web service. Figure 1 illustrates the advantages of using a Data Transfer Object where there is only one call required.

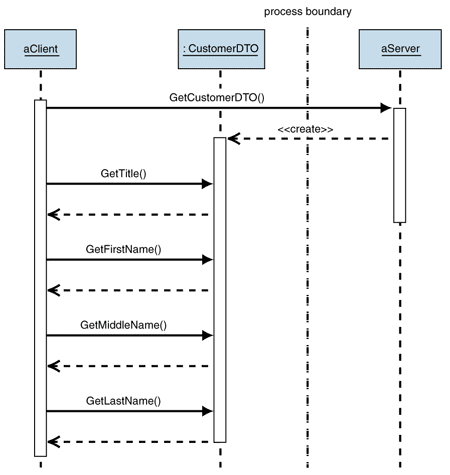


Figure . Demonstrates how Data Transfer Objects (DTO's) help to reduce the number of calls.  
Source: http://msdn.microsoft.com/en-us/library/ff649585.aspx

Making a single web service call with larger amount of information encapsulated is preferred to multiple calls with smaller objects when factors such as network latency and battery lifespan on the mobile device are taken into consideration.

The second group of the data model consists of application’s domain objects which will be mapped directly onto database tables. With the help of Entity Framework, the generic repository and the unit of work patterns as described in chapter 3, it will be possible to manipulate information in the database in terms of business entities without the need to write complex SQL queries.

### 2.1.2 Data Models

This section focuses on the structure of the domain objects which exist within the application’s data model. These object form the foundation of the system and their existence is fundamental the function definitions described later in this chapter.

**User object.**Represents a real-world user entity. User objects represent entities who can register and interact with the system. Users can register, log-in, create journeys, apply for journeys, make modifications, cancel, and add/delete friends and more.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Data Type** | **Range/Length** | **Description** |
| UserId | Integer | 1 - Integer Max. | Unique identifier of the user object. |
| ProfilePicture | ProfilePicture | N/A | Contains user’s profile picture. Please see the ProfilePicture object for more details. |
| UserName | String | 6 - 30 characters | Uniquely identifies user in the system. |
| EmailAddress | String | 6 - 50 characters | Contains user’s email address. |
| First Name | String | 1-50 characters | Contains user’s first name. |
| LastName | String | 1-50 characters | Contains user’s last name. |
| DateOfBirth | DateTime | N/A | Contains user’s date of birth. |
| Gender | Enum | Male, Female | Identifies user’s gender. |
| GCMRegistrationID | String | 1-150 characters | Contains user’s GCM Registration ID assigned to user’s device by Google’s GCM servers. |
| Friends | Collection <User> | N/A | Contains a list of User objects identified as user’s friends. |
| Status | Enum | Online, Offline | Contains user’s current online status. |
| Ratings | Collection <Rating> | N/A | Contains a list of Rating objects representing this user’s ratings left by other users. |
| LastLogon | DateTime | N/A | Contains the date and time of this user’s last logon. |
| PhoneNumber | String | 6-12 characters | Contains user’s phone number. |
| AverageRating | Double | 0 – 5 | Contains user’s current average rating. |
| JourneyTemplates | Collection<JourneyTemplate> | N/A | Contains a list of this user’s Journey Template objects. |
| Notifications | Collection <Notification> | N/A | Contains a list of Notification objects each of which represents a different notification targeted for this user. |
| PrivacySettings | PrivacySettings | N/A | Contains the user’s privacy settings which control the visibility of the users’ profile to the outside world. |

**Journey object.**Represents a journey object which users can advertise, search and send requests for.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Data Type** | **Range/Length** | **Description** |
| Journey Id | Integer | 1 – Integer Max | Unique identifier for the journey object. |
| Driver Id | 1 – Integer Max | 1 – Integer Max | Identifies the user who advertised this journey. |
| GeoAddresses | List<GeoAddresses> | N/A | A collection of GeoAddress objects which contain the journey’s addresses and locations. |
| Departure DateTime | DateTime | N/A | Contains the journey’s departure date and time. |
| Description | String | 0-250 characters | Contains a short message created by the driver. |
| Fee | Double | 0 – Double Max | Contains the fee for this journey specified by the driver. |
| Available Seats | Integer | 1 - 10 | Contains the number of available seats specified by the driver. |
| Passengers | List<User> | 0 – Available Seats | Collection of User objects containing all users who applied for this journey and have been accepted. |
| Pets Allowed | Boolean | True, False | Specifies whether Pets are allowed in this journey. |
| Smokers Allowed | Boolean | True, False | Specifies whether Smokers are allowed in this journey. |
| Private | Boolean | True, False | Specifies whether this journey is only visible to driver’s friends. |
| Vehicle Type | Enum | Private Car, Minivan, Van, Lorry, Motorbike, Other | Contains the vehicle type specified by the user during journey creation. |
| Journey Status | Enum | OK, Expired, Cancelled | Contains the current status of this journey. |
| Creation Date | DateTime | N/A | Contains the date and time when this journey was created. |
| Preferred Payment Method | String | 0 – 250 characters | Contains the preferred payment method for this journey as indicated by the driver. |

**Journey Request** - represents a journey request object that is sent from one user to another asking to join a specific journey.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Data Type** | **Range/Length** | **Description** |
| Journey Request Id | Integer | 1 – Integer Max | Unique identifier of the journey request object. |
| Journey Id | Integer | 1 – Integer Max | Unique identifier of the journey this request applies to. |
| Requesting User Id | Integer | 1 – Integer Max | Identifies user who placed the request. |
| Message | String | 0 – 250 characters | Contains textual message from the requesting user to the driver. |
| Decision | Enum | Accepted, Pending, Denied | Identifies the decision made by the driver regarding this request. |
| Sent Date | Date | N/A | Contains the date on which the request was sent. |
| Decision Date | Date | N/A | Contains the date on which the request was responded to. |

**Friend Request** - represents a friend request object that is sent from one user to another when they wish to add a new friend to their friend’s list.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Data Type** | **Range/Length** | **Description** |
| Friend Request Id | Integer | 1 – Integer Max | Unique identifier of the Friend Request object. |
| Receiving User Id | Integer | 1 – Integer Max | Identifies the user whom the friend request was sent to. |
| Requesting User Id | Integer | 1 – Integer Max | Identifies the requesting user. |
| Message | String | 0 – 250 characters | Contains a textual message included by the From User. |
| Decision | Enum | Pending, Accepted, Denied | Contains the decision made by the ‘Receiving User’ in regards to this request. |
| Sent On Date | DateTime | N/A | Date on which the request was sent. |
| Decision Date | DateTime | N/A | Date on which a decision was made. |

**Session** – represents a session object used in managing the session between the mobile application and the web service.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Data Type** | **Range/Length** | **Description** |
| User Id | Integer | 1 – Integer Max | Contains the identifier of the user whom this session belongs to. |
| Session Id | String | 8 characters | Contains the session’s security token used for authentication. |
| Expiry Date | Date | 0 – 250 characters | Identifies when the session is due to expire. |
| Session Type | Enum | Temporary, Permanent | Identifies whether user has ticked the ‘Remember Me’ option or not. |

**Rating** – represents a rating object used for leaving feedback by a journey passenger.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Data Type** | **Range/Length** | **Description** |
| Rating Id | Integer | 1 – Integer Max | Unique identifier for the Rating object. |
| Receiving User Id | Integer | 1 – Integer Max | Identifies the user whom the rating has been left for. |
| Leaving User Id | Integer | 1 – Integer Max | Identifies the user who submitted the rating. |
| Rating Date | DateTime | N/A | The date the rating was submitted. |
| Score | Integer | 1 – 5 | Score rating left by the From User reflecting their satisfaction. |
| Feedback | String | 0 – 250 char | Textual message to accompany the Score. |

**Profile Picture** – represents an object that contains user’s profile picture saved as an array of bytes. The need to separate user’s profile picture from the main user object came as a result of the following architectural decisions.

* Profile picture should only be downloaded once and stored in Least Recently Used (LRU) memory cache. All subsequent picture requests should use the memory cache to speed up the process.
* Retrieving the user object without the profile picture results in a much smaller HTTP response thus speeding up the object serialisation and deserialization.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Data Type** | **Range/Length** | **Description** |
| Profile Picture Id | Integer | 1 – Integer Max | Unique identifier for the Profile Picture object. |
| Profile Picture Bytes | Byte[] | Byte[0] –  byte[Integer Max] | Stores the picture converted to an array of bytes. |

**Privacy Settings** - represents an object which contains user’s privacy settings. These are used when a user object is retrieved from the web service. Privacy settings determine which items inside user’s profile are visible and at what level, i.e. friends, everyone or private.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Data Type** | **Range/Length** | **Description** |
| Privacy Settings Id | Integer | 1 – Integer Max | Uniquely identifies each Privacy Settings object. |
| Email Privacy Level | Enum | Private, Friends Only, Public | Defines the privacy level applied to user’s email address. |
| Gender Privacy Level | Enum | Private, Friends Only, Public | Defines the privacy level applied to user’s gender information. |
| Date Of Birth Privacy Level | Enum | Pending, Accepted, Denied | Defines the privacy level applied to user’s date of birth. |
| Phone Number Privacy Level | Enum | Private, Friends Only, Public | Defines the privacy level applied to user’s phone number. |
| Rating Privacy Level | Enum | Private, Friends Only, Public | Defines the privacy level applied to user’s ratings. |
| Journeys Privacy Level | Enum | Private, Friends Only, Public | Defines the privacy level applied to user’s journeys. |

**Notification** – represents a notification object. Notification is a system message intended for the user to let them know something that concerns them has taken place. Certain notifications have actions associated with them. For example when a new friend request is received, while others are just textual messages.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Data Type** | **Range/Length** | **Description** |
| Notification Id | Integer | 1 – Integer Max | Unique identifier of the notification object. |
| User | User | N/A | Identifies the user whom this notification belongs to. |
| Notification Message | String | 0 – 1000 characters | Textual message displayed when the notification is downloaded. |
| Read | Boolean | True, False | Indicates whether this notification has already been read by the user. |
| Creation Date. | DateTime | N/A | Contains the date on which the notification was created. |

**Journey Template** – represents an object used to automatically notify the user who created it, when another user advertises a journey which matches the criteria specified in the journey template object. This essentially provides a subscription-like functionality freeing the user from the need to performing manual search.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Data Type** | **Range/Length** | **Description** |
| Journey Template Id | Integer | 1 – Integer Max | Unique identifier for this Journey Template object. |
| Alias | String | 1 – 250 char | Contains the name which the user assigned to this journey template. |
| User Id | Integer | 1 – Integer Max | Identifies the user who created this journey template. |
| Fee | Double | 0 – Double Max | Contains the maximum fee which the user wants the system to consider. |
| Departure Point Radius | Double | 0 – Double Max | Used to include all journeys whose departure points are within this radius. |
| Destination Point Radius | Double | 0 – Double Max | Used to include all journeys whose destination points are within this radius. |
| Pets Allowed | Boolean | True, False | Indicates whether the journey should be allowing pets. |
| Smokers Allowed | Boolean | True, False | Indicates whether the journey should be allowing smokers. |
| Vehicle Type | Enum | Private Car, Minivan,  Van, Lorry, Motorbike,  Other, No Preference | Indicates the preferred type of vehicle which the user would like to travel in. |
| GeoAddresses | Collection <GeoAddress> | N/A | List of addresses which the desired journey must match. |
| Date Allowance | Integer | 1 – Integer Max | Identifies user’s flexibility in regards to the departure date in days. |
| Time Allowance | Integer | 1 – Integer Max | Identifies user’s flexibility in regards to the departure time in hours. |
| Date And Time Of Departure | DateTime | N/A | Contains the preferred date and time of departure. |
| Creation Date | DateTime | N/A | The when the journey template was created. |

**Journey Message** – represents a journey message object, used in multi-user journey chat rooms. Journey chat rooms enable all journey passengers to exchange messages in real-time even if they are not in each other’s friends lists.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Data Type** | **Range/Length** | **Description** |
| Journey Message Id | Integer | 1 – Integer Max | Unique identifier of the journey message object. |
| Journey Id | Integer | 1 – Integer Max | Identifies the journey object which this message belongs to. |
| Sender Id | Integer | 1 – Integer Max | Identifies the sender of the message. |
| Message | String | 1 – 250 char | Contains the textual message sent by the Sender. |
| Sent On Date | DateTime | N/A | Contains the date the message was sent. |

**GeoAddress** – represents a point on map with used for searching and advertising journeys.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Data Type** | **Range/Length** | **Description** |
| GeoAddress Id | Integer | 1 – Integer Max | Unique identifier of the GeoAddress object. |
| Address Line | String | 1 – 250 char | Contains the human readable address. |
| Latitude | Double | -90 - +90 | Contains the latitude of this address. |
| Longitude | Double | -180 - +180 | Contains the longitude of this address. |

**Chat Message** – represents an instant message object, used during real-time conversation between two users.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Data Type** | **Range/Length** | **Description** |
| Chat Message Id | Integer | 1 – Integer Max | Unique identifier of the chat message object. |
| Sender Id | Integer | 1 – Integer Max | Identifies the sender of this message. |
| Recipient Id | Integer | 1 – Integer Max | Identifies the recipient of this message. |
| Message | String | 1-250 characters | Contains the textual message sent by the recipient. |
| Sent On Date | DateTime | N/A | Contains the date on which the message was sent. |
| Read | Boolean | True, False | Identifies whether this message has been read by the recipient. |

## 2.2 Function Definitions

This section focuses on the functions that have derived from the requirements from the ‘Requirements’ section. Below is a list of distinct functions which were identified across the system:

* User Service
* Register new user
* Manual user login
* Auto user login
* Get user by Id
* Update user profile
* Journey Service
* Advertise a new journey
* Modify existing journey
* Get journey by id.
* Get list of passengers for journey.
* Get all requests for journey.
* Journey Template Service
* Create a new journey template
* Modify existing journey template
* Get journey template by id
* Get user’s journey templates
* Delete journey template
* Journey Request Service
* Send a new journey request
* Process decision for a journey request
* Get journey request by id
* Journey Request Service
* Send a new journey request
* Process decision for a request
* Get journey request by id
* Session Manager
* Generate a new session
* Validate user’s session
* Invalidate user’s session
* Notification Manager
* Send GCM tickle
* Send instant message
* Send notification
* Search Service
* Search for journeys

A very small sample of the function definitions associated with each of the above functions has been included below. For a complete list of function definitions, please refer to the appendices.

|  |  |
| --- | --- |
| **Function name: Registration** | |
| **Objective** | Users should be able to register in order to access the system using their account. |
| **Parameters** | User Model – please refer to the User object in chapter 2 for more details. |
| **Action** | 1. Users enters their desired username, email address and password of their choice twice for confirmation. 2. User clicks the “Create account button” 3. Android Application sends request to the WCF web service to register a new account. 4. WCF web service validates the parameters provided by the user. 5. WCF web service creates new User and adds it to the database to the Users table. 6. The security module encrypts user’s password and stores it in a separate table. 7. WCF sends a reply to the Android client informing of a successful registration. 8. Android application logs the user in automatically. |
| **Requirements** | 1 |
| **Pre-conditions** | * User must install the application on their Android device. |
| **Post-conditions** | * New user will be created from the parameters supplied will be created and added to the database. |
| **Error conditions** | * User attempts to register with username already in use. * User attempts to register with email already in use. * User attempts to register with blank username. * User attempts to register with blank email address. * User attempts to register with an invalid email address. * User attempts to register with blank password. * User attempts to register with passwords that do not match. * Users fails to provide a password that’s at least 6 characters long. |

|  |  |
| --- | --- |
| **Create Journey** | |
| **Objective** | Users should be able to advertise journeys. |
| **Parameters** | Journey Model – please refer to theJjourney object in chapter 2 for more details. |
| **Action** | 1. User specifies departure and destination points for their new journey, as well as any optional waypoints. 2. User fills in additional journey information to populate the Journey Model from the Parameters section. This information includes:    * Departure date.    * Departure time.    * Private    * Allowed Pets.    * Allowed Smokers.    * Vehicle Type.    * Available Seats.    * Fee.    * Additional comments. 3. User confirms their decision to advertise the new journey. 4. WCF web service validates the journey model supplied by the user. 5. WCF saves new journey in the database making it available for other users to see. |
| **Requirements** | 4 |
| **Pre-conditions** | * User must be logged in. |
| **Post-conditions** | * A new journey object will constructed from the above parameters and added to the database. * The new journey will immediately become available for other users to search for. * The user who offered the journey will be associated with the journey as the driver and will be able to manage it through the journey management activity. |
| **Error conditions** | * User attempts to advertise a journey without specifying any of the following parameters: * Departure date * Departure time * Vehicle type * Fee * Departure address * Destination address |

# Design

This chapter focuses on the system architecture and the user-interface design process giving a thorough explanation for the decisions that have been made in the initial stages of the project.

## User Interface Design

The initial user interface designs have been produced using an online wire-framing utility - balsamiq, (<http://webdemo.balsamiq.com/>). It’s important to note however, that the initial user interface sketches were used as a guideline for development of the final designs featuring a much higher level of sophistication and increased user-friendliness.

### 3.1.1 Mobile Device Considerations.

During the development of the user interfaces for the Android application, it was extremely important at all times, to keep in mind that the process of developing user-interfaces for mobile devices is significantly different from development of user interfaces for standard desktop applications. The rules and principles learned during development of desktop user interfaces cannot be applied to mobile devices. It is for these reasons, that the user interfaces in the Android app have been designed to convey the information clearly and effectively while making use of the available screen space with the help of touch-friendly controls.

#### 3.1.1.1 User interface elements.

With the multitude of mobile devices currently on the market, all featuring different screen resolutions, densities, sizes as well as aspect ratios, the development of user interface elements such as buttons or textboxes must be performed in terms of screen proportions. This means that a button should be set to occupy 80% of screen width rather than specifying fixed values to ensure proper scalability and positioning. This is because while a button that’s 200 pixels wide might look perfect on a certain device, it will most likely look differently on a device with a different screen size thus ruining the user-experience and in extreme scenarios rendering the button unusable. As an example, please see figure two which illustrates how the above principle can be applied to a real user interface. The top “Show Profile” button, has been configured to occupy 100% of the available screen width while the two buttons at the bottom, the “Accept” and “Deny” buttons, share the 100% of the screen width between them, where each occupies exactly 50% of the screen width.



Figure illustrates how user-interface scaling in Android works.

#### 3.1.1.2 Expert vs novice users.

Users are often divided into distinct groups with different intentions and levels of expertise. Novice users, are much more likely to make touch errors and move slower throughout the application. Expert users on the other hand, tend to memorize the layout and functions of the applications thus being able to navigate much more quickly. The application’s user interface has been designed to be as intuitive and informative as possible essentially guiding the user through the application as they navigate from one screen to another. With such design in place, novice users will be able to learn and navigate through the application quickly while the expert users will not be distracted by the subtle hints that have been put in place to guide the novice users. Moreover, the Android application provides features such as input validation and help content which the novice users will find extremely helpful when finding their way through the application.

#### 3.1.1.3 User-Interface location and accounting for motion

Another very important factor which has an influence on the process of user-interface development is the way users hold and interact with their mobile devices. Most users hold their mobile devices in one of a few ways, with either their thumb or their index finger acting as the primary means of providing an input.

This means that reaching user interface controls in one corner of the screen might be easier and less intrusive than reaching the controls in the opposite corner depending on how the device is held by the user. As an example, please consider figure 3 which illustrates the ease of interacting with a user interface element located in a certain area for a right-handed user using their thumb as their primary device.



Figure user-interface ease of access map for a right handed user.  
Source: http://venturebeat.com/2013/04/08/5-tips-for-creating-great-mobile-app-user-interfaces/

In order for the application to provide optimal user-experience for all groups of users, majority of user interface elements have been positioned near the centre of the screen either at the bottom, where they are easily reachable with user’s thumb, or near the top where the users can interact with them easily using their index finger. In addition, all user interface elements have been positioned away from the edges of the screen providing a certain amount of margin thus making it easier by the user to target a specific user control and successfully interact with it on their first attempt as figure 3 below shows.

### 3.1.2 Android Application.

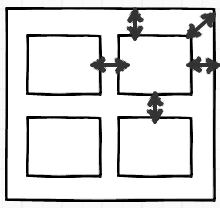


Figure margin between the edges of the screen and the user-interface elements

While designing the theme for the Android application, it was extremely important to keep in mind the guidelines for mobile application user interface design as described in section 3.1.

#### 3.1.2.1 Initial sketches.

One very important factor to consider is that the interaction between the user and the mobile application often takes place on the move meaning user is able to fully concentrate on the information displayed on the screen. This means that the various user interface elements must be designed and positioned with touch-friendliness and usability in mind while still being able to convey the information in the best possible way.

The purpose and meaning of each individual textbox and button control must be clear right the second it is seen. This means that each user interface component visible on the screen has a very specific task.

Figure 2 demonstrates a small subset of the initial sketches for the user’s for some of the activities. Please refer to the appendix for a complete list of user interface sketches.   
3.1.2.2.2 Colour Theme



Figure illustrates the initial user interface sketches. Starting from the left, activity used when searching for journeys, activity displaying user's journeys and user home activity.

The application’s colour theme is based on a small set of colours should be used for consistency and clarity purposes. After much trial and error, the following three base colours have been selected to serve as a foundation for the Application’s theme.

* Icons - #32B4E4. (Light blue)
* Buttons - #80151515 (Dark grey with 50% transparency applied.)
* Buttons (Pressed) #807E7E7E (Light grey with 50% transparency applied.)
* Text - #FFFFFF (White)

#### 3.1.2.3 Final design.

Figure 2 demonstrates the finalised design of the application’s home activity with the above colour theme applied.

This particular combination of colours reduces the eye-strain due to domination of dark shades of grey which have been proven by various studies to have less negative effects on the user’s eyes.

The final design offers simple yet eye-catching user interface. It provides quick access to the most important functions of the application via touch-friendly user interface components which present the user with the relevant information in a clear way without unnecessary clutter. Figure 3 demonstrates how the initial sketches from figure 2 have been developed into the final designs. For a complete list of screenshots from the Android application, please refer to the appendices. //TODO – Insert appendix reference.

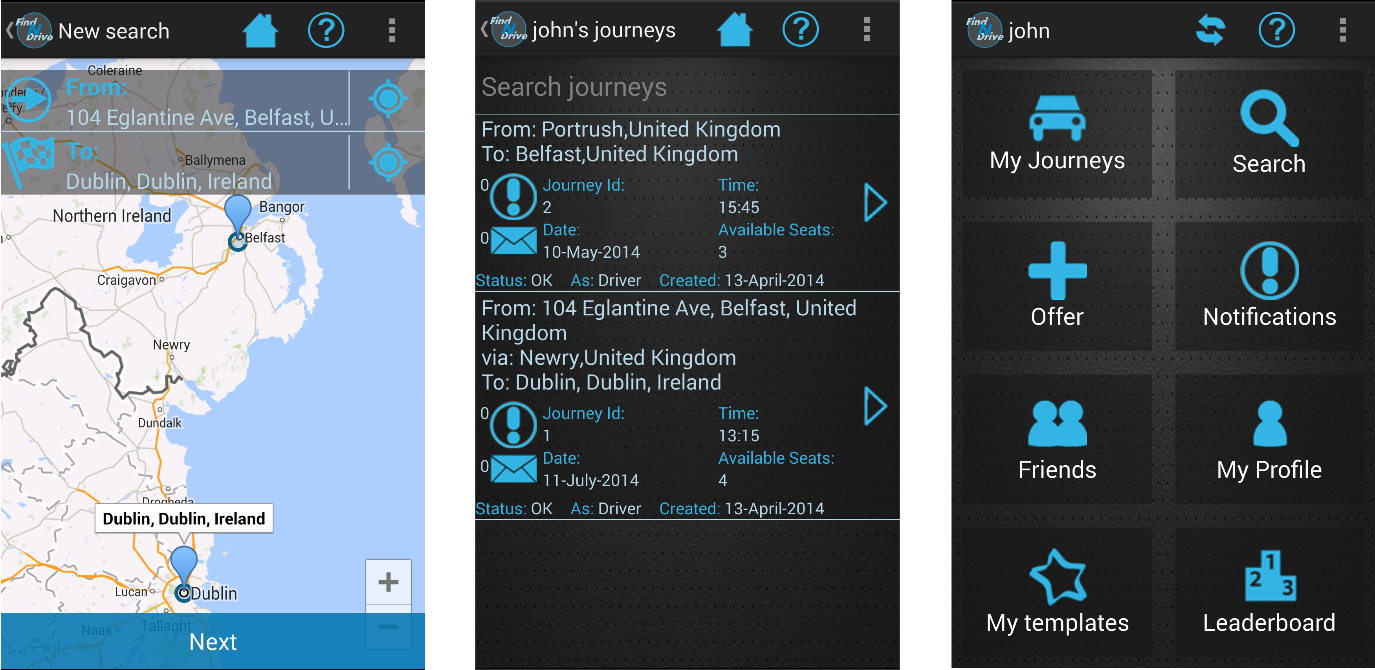


Figure 6 demonstrates the final design of the initial user interface sketches shown in figure 2.

#### 3.1.2.4 Navigation

Navigation and screen management in Android is based on a simple yet very effective principle. Screens in Android, which from here on will be referred to as activities, are pushed onto a stack as the user navigates from one activity to another, and popped off it when the user presses the back button.



Figure . Figure 4. Android's activity management using a stack-like approach.   
Source: https://developer.android.com/guide/components/tasks-and-back-stack.html

The Android application features a stack-based navigation system which is compatible with Android’s native way of handling activity transitions. After the application is launched the user, is greeted with the login activity which prompts them to enter their username and password. On successful login, user is transferred to the home screen as shown in figure 3, from which they can access all other functions of the application. Please see figure 6 for a complete navigation map.

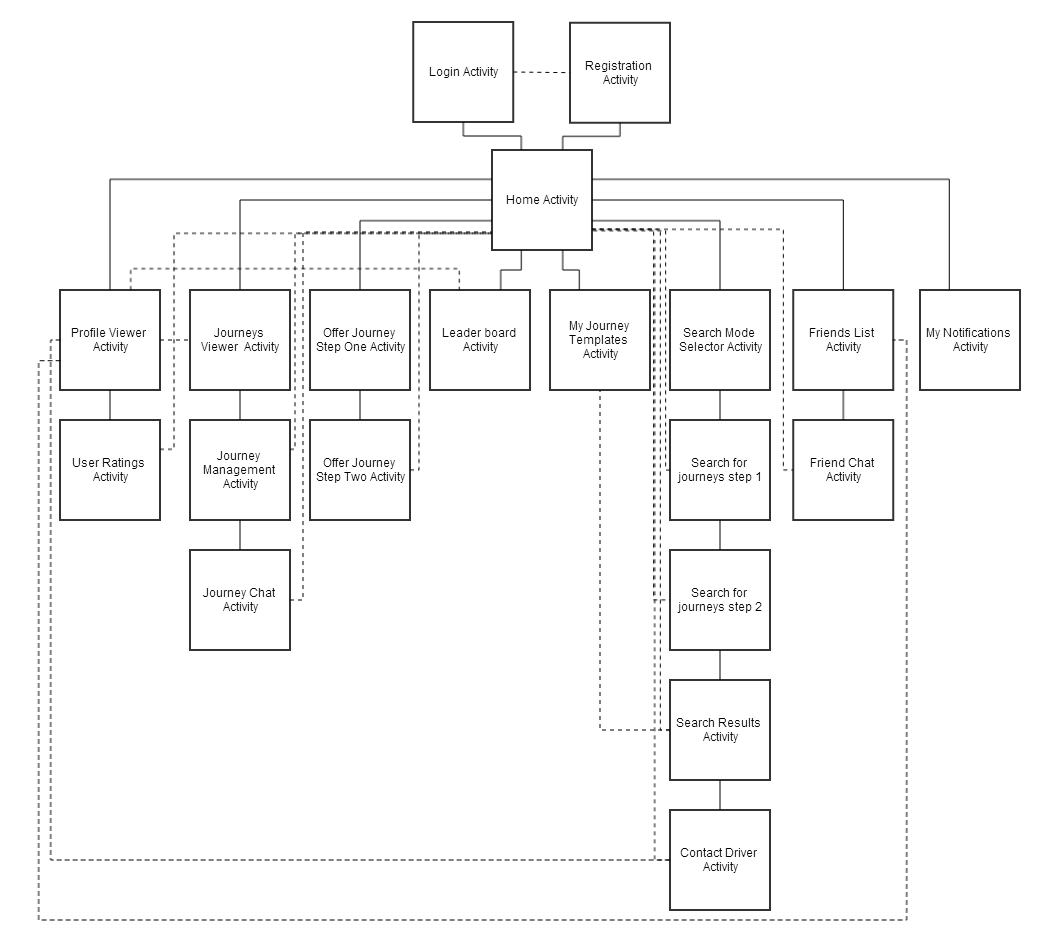


Figure illustrates the Application navigation map.

Each of the activities contains a ‘Home’ button in the top section allowing the user to quickly cancel their current task and return back to the home activity. Moving back through the activities stack can be performed in two ways, either by pressing the Android’s built-in back button or by clicking the back button located in the action bar in the top left corner.

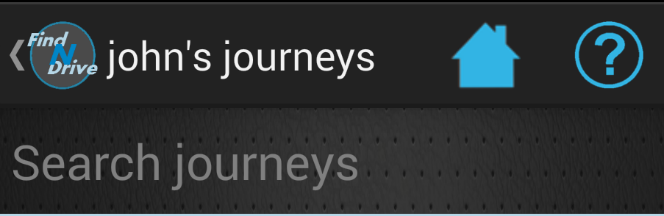


Figure Back button built into the action-bar

#### 3.1.2.5 User Interface Elements

The applications’ user interface are constructed from the following types of components:

**Layouts**

All Android user interface elements must be placed inside their parent (or root) element. These parent elements are known as layouts. Their primary role is defining the visual structure of a user interface, such as the UI for an activity or app widget. Different types of layouts arrange the content in a different fashion. The application makes use primary of the LinearLayout, which arranges content in a single row or column, and RelativeLayout where the position of the children can be described in relation to each other or to the parent.

**ListViews**

ListView is a view group components that displays a list of scrollable items. The items are mapped onto the ListView using an adapter. An adapter converts the data source such as an array into a view that’s placed into the list allowing for creation of very complex views.

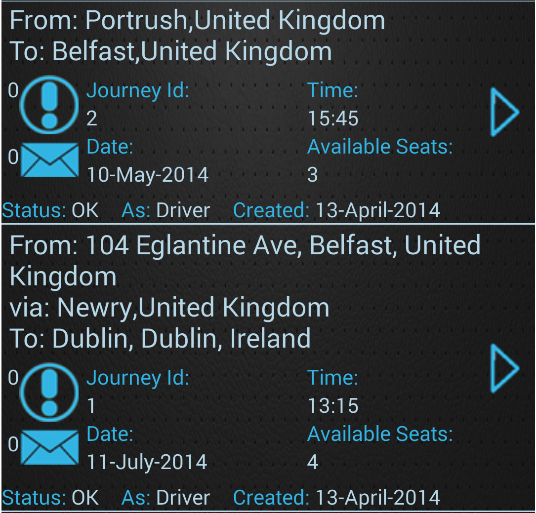


Figure ListViews are very flexible and the information to be displayed can be arranged in a variety of different ways.

ListViews are used quite extensively in the application due to their flexibility. The content can be displayed in a variety of different ways and the ListView itself has appropriate event handlers to respond when user clicks on an individual item.

Figure 7 illustrates how a ListView can be used to display information about a complex entity such as the journey object in a clean and easy to read fashion.

**TextViews**

TextViews are used to display textual information. They offer full CSS-like styling to enable creation of attractive and visually appealing content.

**EditTexts**

EditTexts, very much like TextViews are used to display textual information. The main difference however, is that they allow the user to edit the text inside them. EditTexts are used primary for capturing user’s text input.

**Buttons**

Button is a push-button widget which can be clicked or pressed by the user to perform an action. Android button provide the standard-button functionality, allowing click, touch and many other events to be captured and handled.

## System Software Design.

### 3.2.1 Top-Level Components.

The system consists of a number of components. These are as follows:

* Database
* WCF Web Service
* Android Application
* Google Cloud Messaging Servers (GCM)
* Google Maps Servers

Figure 5 illustrates network diagram of the entire system. With respect to the principles of the n-tier design pattern as explained in the next section, each of the components exists in a separate layer. One of the biggest advantages of this approach is the redundancy and improved testing which this separation provides. Calls to the web service can be routed either through the development or the production server and target the same database which resides in its own database engine provided by Amazon AWS.

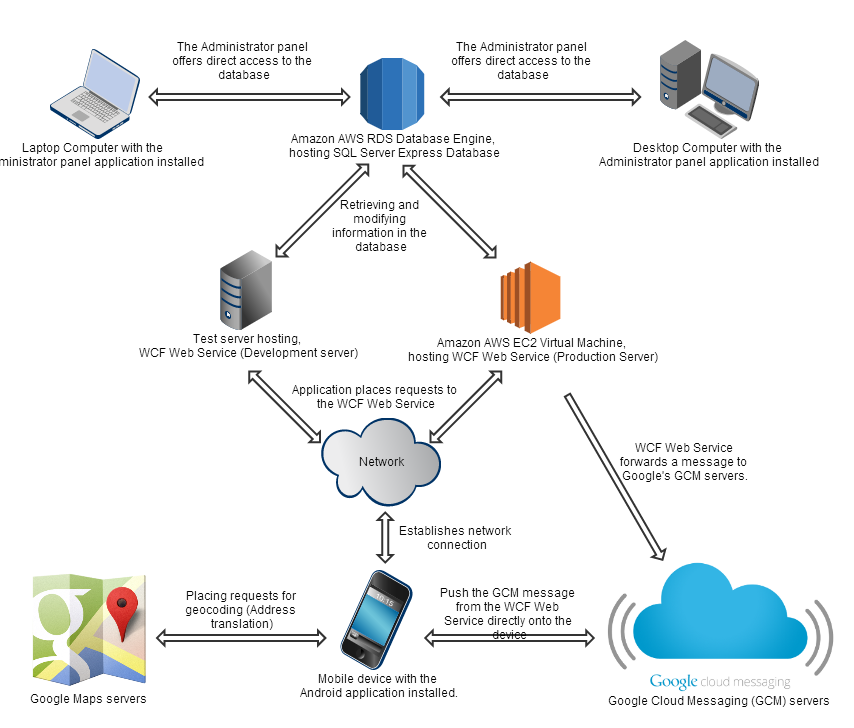


Figure . Network diagram illustrating high-level architecture of the entire system.

### 3.2.2 Multi-tier architecture.

The entire solution has been designed with the best practices of Software Engineering in mind. One of the most important decisions made in the initial stages of the project was to design the entire solution using the multi-tier architecture, often referred to as n-tier, supporting the concepts of Service Oriented Architecture (SOA) offered via the WCF service layer and through the application of SOLID. Many existing Gang of Four patterns, such as the Strategy pattern and Command pattern, and newer patterns such as the Repository and Unit of Work patterns have been effectively applied throughout the development process to provide for a decoupled, decentralised, functionally cohesive system. In particular an emphasis on the Interface Segregation Principle was applied, which allowed for improved testability during both Unit tests and Acceptance tests due to the clear separation of responsibilities.

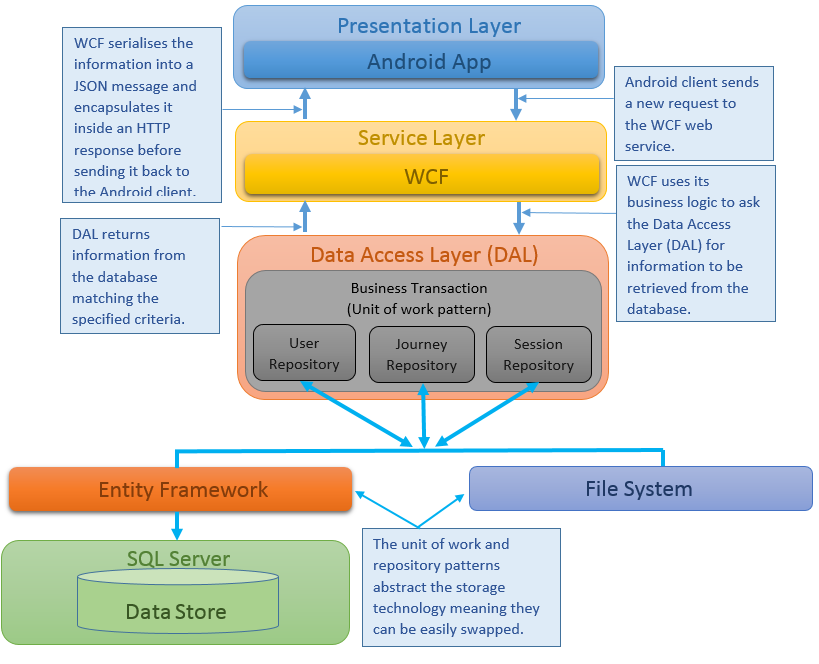


Figure illustrates the application's multi-tier design approach.

The decoupling between individual tiers makes the application highly scalable with clear separation of concerns. It also provides for better security and fault tolerance since each layer has a built in mechanism for error handling and data validation. Figure //TODO insert figure no. represents a top-level view of the entire application with clear distinction between individual layers.

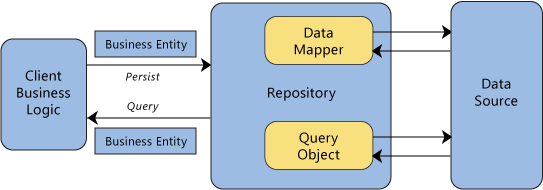
### 3.2.3 Data Access Layer

The data access layer will be designed to provide access to the data source through an abstraction layer. The main responsibility of this abstraction layer will be to provide seamless access to the data source independent of its underlying technology. It will also provide a set of data manipulation methods which will act as a means of retrieving the data from the data source without the need to write complex SQL queries.

* + - 1. Generic Repository Patterns and unit of work patterns.

To achieve the best possible level of abstraction, the generic repository and unit of work patterns will be implemented. Both patterns are intended to create an abstraction layer between the data access layer and the business logic layer of an application to provide for seamless and easy CRUD operations.

The repository pattern will be responsible for retrieving the data from the data source and mapping it onto the entity model from the business logic layer. It works by adding a separation layer between the data and domain layers of an application and provides access to the data source via a number of centralized repositories. This eliminates all code duplication inside the data access layer and provides a flexible architecture that can be adapted as the overall design of the application evolves.



The unit of work pattern on the other hand, performs two very important tasks. First it maintains in-memory updates and second it sends these in-memory updates as one transaction to the database.

Each time an entity is retrieved from the data source using the repository pattern, the unit of work maintains the entity’s current state in the memory. This allows to make multiple changes to the entity before finally committing them in a single atomic transaction essentially persisting them in the data source. Another big advantage of the pattern is its reusability. The code responsible for handling of the transactions is generic making it reusable with all types of entities thus also avoiding unnecessary duplicated code.

//TODO - insert diagram showing the unit of work pattern.

#### 3.2.3.2 Database Design

The application’s database will be designed using Entity Framework’s code first approach. The code first approach uses application’s model objects to generate the entire database without the need to ever open the database designer.

The code-first approach offers a number of advantages over the traditional database design. One of them is the more code-centric approach in defining of the database schema. This is due to Entity Framework’s ability to generate the database model from ‘plain old classes’ with no base classes required. As an example, consider the following class representing the ‘User’ object.

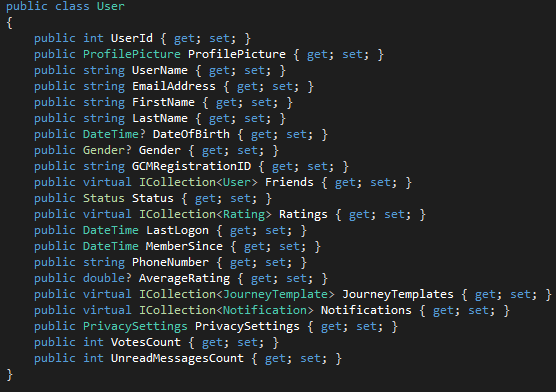


Figure illustrates an example of a typical C# class file used to generate database schema with Entity Framework's code first approach.

After executing the ‘update-database’ command, in Visual Studio Package Manager’s console, the database is automatically generated with the Users table already present and with the following schema:

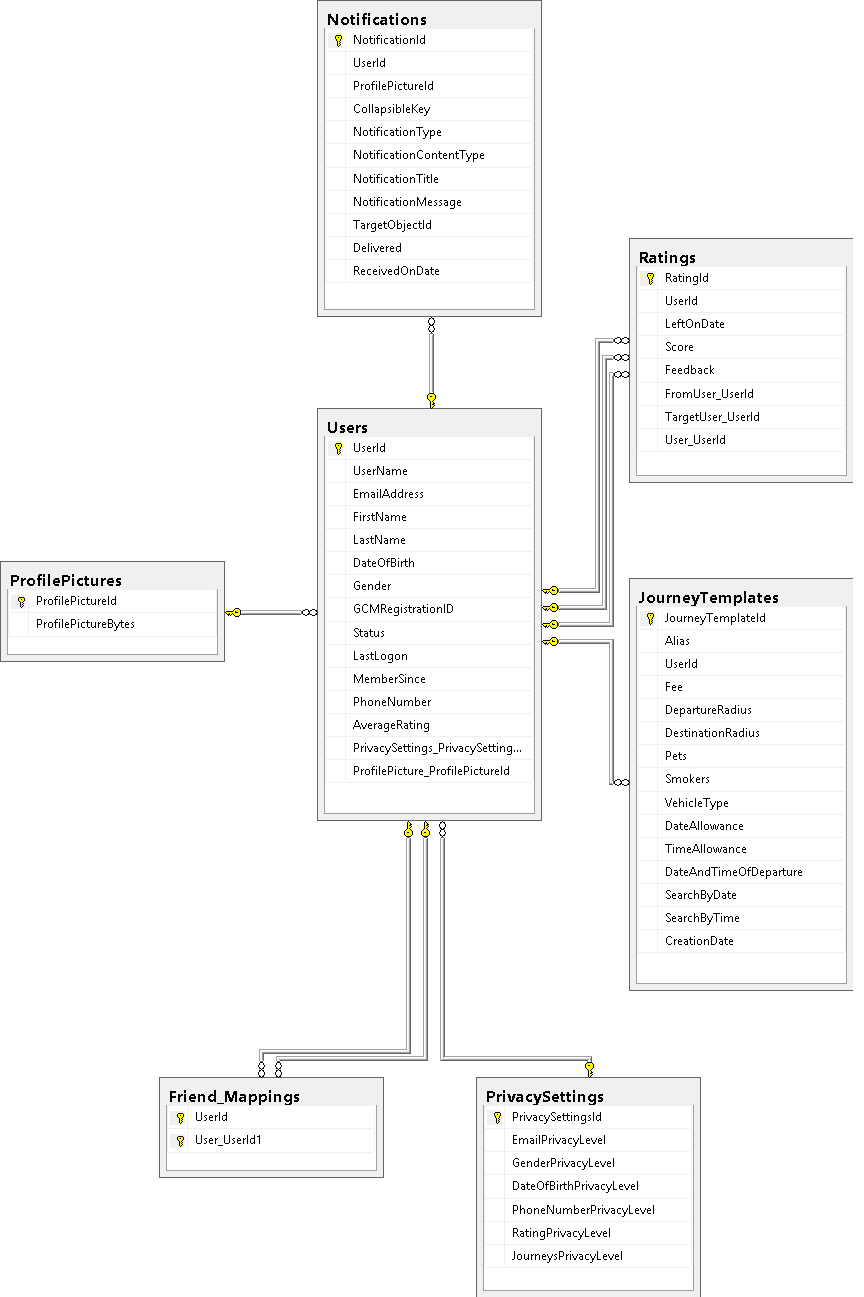


Figure Illustrates the schema generated from the User class as shown in Figure 7

### 3.2.4 WCF Web Service

The WCF Web Service has served as a foundation for the Service Oriented Architecture (SOA) as described in section 3.2.2. In order implement the SOA architecture successfully, the WCF service layer has been divided into multiple functionally cohesive services independent of one another. Each of the services focused on a specific set of tasks and only provided functionality for the business logic for which it was intended. For example, the User Service within the service layer is only responsible for a set of task related to management of user accounts such as logging in, registration or retrieval of user’s information.

As an example, please consider figure 11, which illustrates the interfaces from which individual services will be implemented. These interfaces only contain the signatures of the methods that will be exposed by each of the services. For a complete list of interfaces, please refer to the appendices. //TODO.

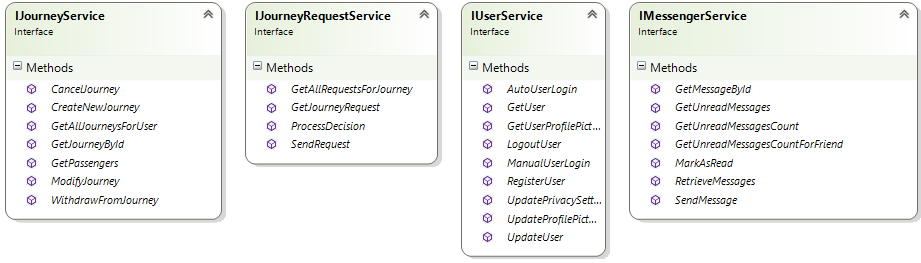


Figure illustrates the interfaces which the WCF services will be implemented.

One of the main benefits of this architectural decision is horizontal scalability, which otherwise would not be easily possible if the WCF service logic was not separated. Horizontal scalability implies that each of the WCF services used within the application could potentially be deployed separately on a different machine balancing the workload. It would also eliminate the need for a high-end system that will be able to host all services at once. This decision would allow for potential fail over mechanisms to be used, and other benefits such as automatic load balancing within contentious areas of the application could be harnessed.

In order to provide a consistent API throughout the numerous exposed service contracts, a consistent and generic ServiceResponse was created. This Service Response object acts as a secure wrapper for data sent back from the WCF service to the client, which is used to indicate whether a particular call resulted in a success or failure. This proved to provide a far richer API than the alternative of throwing Exceptions. When a request is made to the WCF service, each response from the service is wrapped inside a Service Response objects which contains the following information:

- ServiceResponseCode: Indicates whether the service call resulted in success or failure. As an example, the user service might return a failure as a service response code if the user provides incorrect login credentials. These initial status codes code be expanded upon within future sprints. For instance the client could then use this information to potentially invoke an operation retry, or provide some sort of business-level compensation logic, for instance if a business-level transaction was being performed.

- ErrorMessages: An optional list of error messages that contain the useful information as to why an operation did not succeed (For instance validation reasons). Within the current architecture, this information was used within the presentation layer to provide the user with extra information as to why an action did not succeed.

- Result: This property was provided to operations which were not defined as ‘fire and forget’, that is to say an operation which provided a return value. The result was of type T, where the type of T depends on the WCF service called, i.e. Project Service might return a result type of ‘Project’

Additionally Data Transfer Objects (DTOs) were provided as a means of decoupling a complex domain model from a specific operation, as a domain object could potentially contain many irrelevant fields to the required operation. This design decision lead to improved testability, and aided in providing a service layer which could be shared potentially by numerous different projects, and under various contexts as the services were not domain-specific, due to the abstraction of DTOs. This decision also benefited a decoupling such that any future updates to domain models could be made without breaking existing API contracts, and therefore avoiding possible regression issues.

### 3.2.5 Security

Being of a paramount concern, the security aspect of the project has been approached very seriously from the initial stages. Due to sensitive nature of data being handled by the application such as user’s personal detail, it was extremely important to implement appropriate security mechanisms that prevent unauthorised parties from gaining access to the information stored in the database.

#### 3.2.5.1 Transport Security

Transport security will be achieved with HTTPS. Each the Android application communicates with the WCF web service, a secure channel over an insecure network is created to prevent eavesdroppers and man-in-the-middle attack with the help of a self-generated SSL certificate which the Android application has been configured to trust. It’s important to note also, that the WCF web service has been configured to only accept incoming requests coming through the HTTPS protocol, any attempt to contact the server using an unsecured connection is automatically rejected.

#### 3.2.5.2 Session Management

A session management mechanism will allow the user to perform a single manual log-in using their username and password and stay logged in for a period of up to two weeks. Any subsequent application launches will use the security token which was received from the WCF web service on the first manual login to authenticate user’s request without the need to exchange user’s username and password.

The security token used to authenticate user’s request will be injected as one of the headers into the HTTP request which the WCF web service will then extract and perform the validation procedures by comparing the data in the security token against information stored in the database.

To ensure user accounts cannot be spoofed by copying another user’s security token, part of the token will be stored on user’s Android device’s persistent storage, while the second part will be generated and added to the security token dynamically and will consist of a set of device identifiers unique to the user’s device. For detailed implementation of the session management mechanism, please refer to chapter 4. //TODO – add reference.

### 3.2.6 Google Cloud Messaging (GCM)

One of the biggest limitations of the Service Oriented Architecture (SOA) is the unidirectional communication channel between the client and the server. The only way for the server to pass new information onto the client is by waiting for the client to place a new request. The client however, might be configured to poll the server to check if new information is available at regular intervals, thus negatively impacting the device’s battery lifespan, increasing unnecessary server load and causing unnecessary traffic.

To solve this problem, bi-directional communication pattern should be implemented whereby a server has a means of contacting the client directly to either inform them that new data is awaiting on the server and it is time to synchronise its state, or by passing the data directly onto the client directly.

#### 3.2.6.1 Google Cloud Messaging Architecture.

Google Cloud Messaging (GCM) offers a solution to the problem by enabling bi-directional communication between the client and the server with the help of Google’s own GCM servers. The GCM system works by maintaining an open connection between the Android client and Google’s GCM servers. The WCF web service acts as a proxy service by forwarding messages onto GCM servers which in turn push the message directly onto the user’s Android device. Figure 3 demonstrates the architectural overview of the Google’s GCM architecture. Figure 4 on the other hand demonstrates a more detailed data flow between the WCF web service, GCM servers and the Android device.



Figure . Google Cloud Messaging (GCM) architecture.

### 3.2.7 Google Maps

One of the most important features of the application, as described in section 1.4 is the ability to perform a location-aware search. To achieve this, the system will implement Google Maps make extensive use of its Android V2 API. Google Maps will provide the application with means of address geocoding. Geocoding is the process of translating the address entered by the user in textual form into latitude and longitude locations. The process, used in conjunction with the Haversine formula will be used as a basis of the location-aware search that finds journeys departing from places nearby.

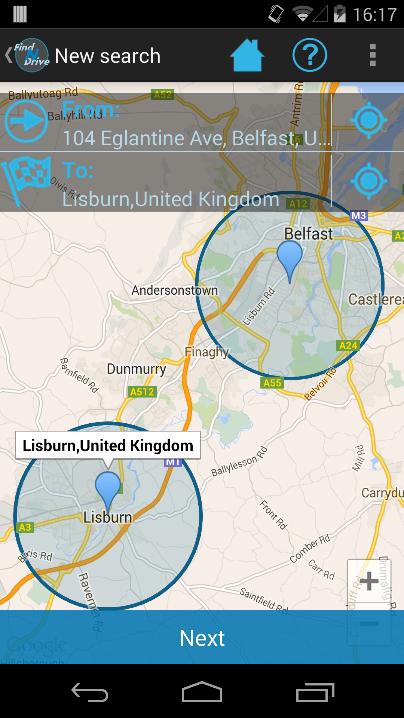


Figure . The search function making use of Google Maps to display departure and destination points of a journey that the user is searching for.

The second aspect in which Google Maps will greatly benefit the project is the increased usability and improved user experience. Addresses entered by the user will be automatically translated into their respective latitude and longitude values. The application will then use those values to show the address on the map in form of a marker. Consider figure 4 as an example, where the user searching for a journey uses the user interface elements overlaying the map to enter the desired addresses which are converted into latitude and longitude values and then shown on the map.

The map will be able to accommodate multiple markers for a visual representation of a journey starting at one point on the map and possibly going through a number of different waypoints before reaching its destination.

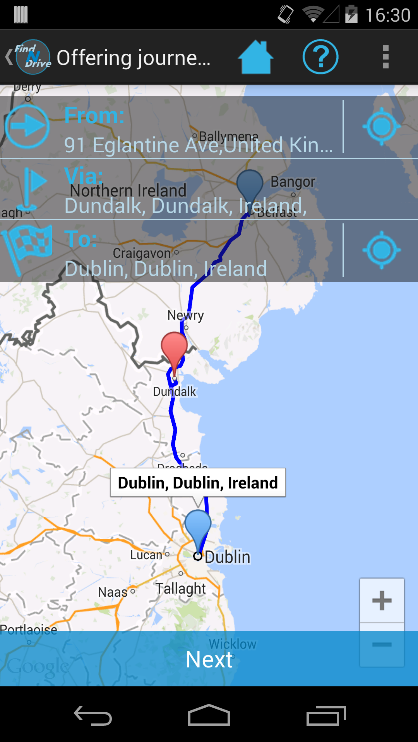


Figure . The process of offering a journey.

As an example of this, please consider figure 5, which demonstrates the process of offering a journey allowing the user to specify start and end points of a journey, as well as optional waypoints in-between.

To further enhance the visual representation of a journey, the application will retrieve a driving route and display it on the map.