**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. 5](#_Toc386090507)

[1.1 Problem description. 5](#_Toc386090508)

[1.2 Problems with current approach 5](#_Toc386090509)

[1.3 Solution 6](#_Toc386090510)

[1.4 Advantages of the new approach 7](#_Toc386090511)

[1.5 Goals 7](#_Toc386090512)

[1.6 Requirements 7](#_Toc386090513)

[Must haves: 8](#_Toc386090514)

[Should haves: 8](#_Toc386090515)

[Could haves: 8](#_Toc386090516)

[Would like to haves: 8](#_Toc386090517)

[1.7 Requirement Modifications. 9](#_Toc386090518)

[1.8 Hardware & Software Platform. 10](#_Toc386090519)

[1.9 Development Languages & Technologies used. 10](#_Toc386090520)

[2. Specification 11](#_Toc386090521)

[2.1 Data Model 11](#_Toc386090522)

[2.1.1 Data Model Architecture 11](#_Toc386090523)

[2.1.2 Data Models 12](#_Toc386090524)

[2.2 Function Definitions 13](#_Toc386090525)

[3. Design 16](#_Toc386090526)

[3.1 User Interface Design 16](#_Toc386090527)

[3.1.1 Mobile Device Considerations. 16](#_Toc386090528)

[3.1.2 Android Application. 17](#_Toc386090529)

[3.2 System Software Design. 22](#_Toc386090530)

[3.2.1 Top-Level Components. 22](#_Toc386090531)

[3.2.2 Multi-tier architecture. 23](#_Toc386090532)

[3.2.3 Data Access Layer 24](#_Toc386090533)

[3.2.4 WCF Web Service 25](#_Toc386090534)

[3.2.5 Security 26](#_Toc386090535)

[3.2.6 Google Cloud Messaging (GCM) 26](#_Toc386090536)

[3.2.7 Google Maps 27](#_Toc386090537)

[4. Implementation 29](#_Toc386090538)

[4.1 Development Platforms and Libraries Used. 29](#_Toc386090539)

[4.1.1 Mobile Application 29](#_Toc386090540)

[4.1.2 Web Service 30](#_Toc386090541)

[4.1.3 Data Access Layer & Database Technology 30](#_Toc386090542)

[4.1.4 Source Control 30](#_Toc386090543)

[4.2 Development Process 31](#_Toc386090544)

[4.2.1 Coding Approach 31](#_Toc386090545)

[4.2.1 Coding Style 31](#_Toc386090546)

[4.2.1.1 Android Application Coding Style 31](#_Toc386090547)

[4.2.1.2 Web Service Coding Style 31](#_Toc386090548)

[4.3 Android Application Implementation 32](#_Toc386090549)

[4.3.1 Setting up the environment 32](#_Toc386090550)

[4.3.2 Implementing Activities 32](#_Toc386090551)

[4.3.3 Developing User Interfaces 33](#_Toc386090552)

[4.3.4 Network connectivity & Multi-threading 34](#_Toc386090553)

[4.3.5 Data Serialisation/Deserialization 35](#_Toc386090554)

[4.3.6 Persisting data 35](#_Toc386090555)

[4.3.7 Client session management 36](#_Toc386090556)

[4.3.8 Google Maps, GPS, Geocoding. 36](#_Toc386090557)

[4.3.9 Working with Images and LRU image cache. 38](#_Toc386090558)

[4.3.10 Input Validation 38](#_Toc386090559)

[4.3.11 Sending Broadcasts 39](#_Toc386090560)

[4.4 WCF Web Service Implementation 39](#_Toc386090561)

[4.4.1 Setting up the environment. 39](#_Toc386090562)

[4.4.2 Implementing Service & Operation Contracts 39](#_Toc386090563)

[4.4.3 Implementing Data Access Layer 40](#_Toc386090564)

[4.4.4 Location-aware search processor 42](#_Toc386090565)

[4.4.5 Deploying Services to AWS and hosting in IIS. 43](#_Toc386090566)

[4.5 Cross-Platform components 44](#_Toc386090567)

[4.5.1 Session Management 44](#_Toc386090568)

[4.5.2 Notification System 46](#_Toc386090569)

[4.6 Test Strategy and Test Suite 49](#_Toc386090570)

[4.6.1 Unit Testing 49](#_Toc386090571)

[4.6.2 User-Interface Testing 49](#_Toc386090572)

[4.6.3 Integration/Acceptance Testing 49](#_Toc386090573)

[4.6.4 Performance Testing 50](#_Toc386090574)

[5 Conclusion 51](#_Toc386090575)

[Appendix 52](#_Toc386090576)

*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.

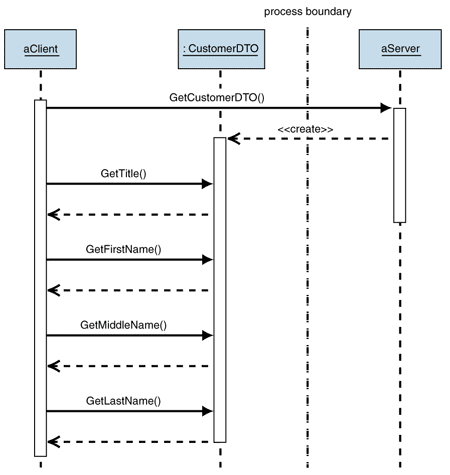


Figure . Demonstrates how Data Transfer Objects (DTO's) help to reduce the number of calls.  
Source: http://msdn.microsoft.com/en-us/library/ff649585.aspx

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.

**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. |

## 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.

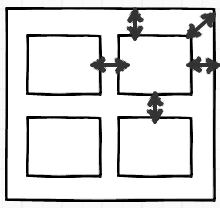


Figure margin between the edges of the screen and the user-interface elements

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.

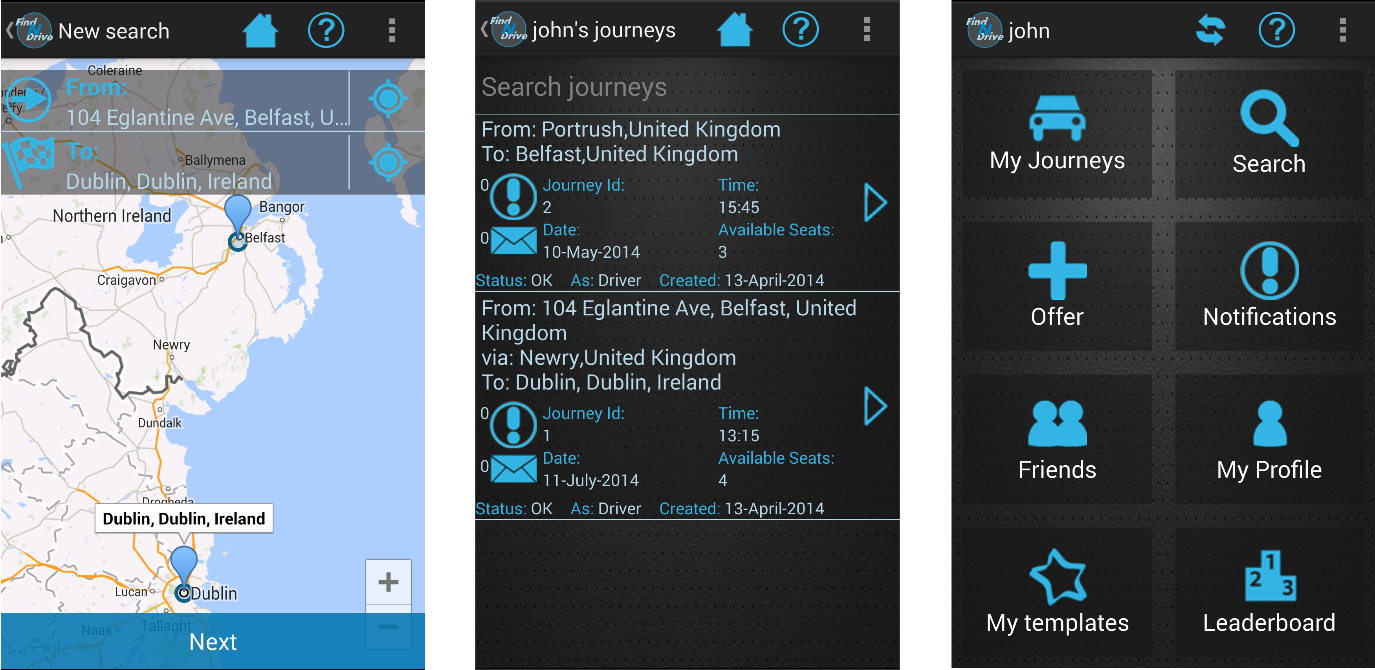


Figure 6 demonstrates the final design of the initial user interface sketches shown in figure 2.

#### 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.

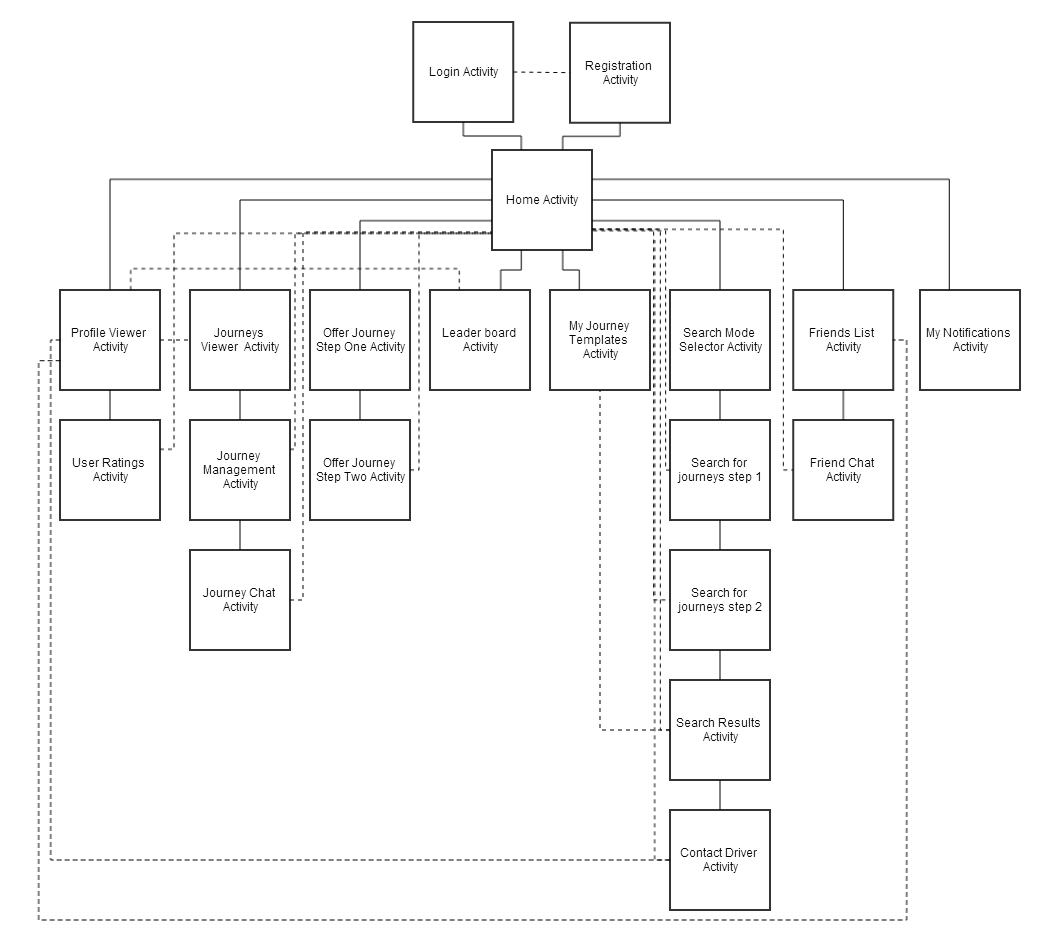


Figure illustrates the Application navigation map.

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.

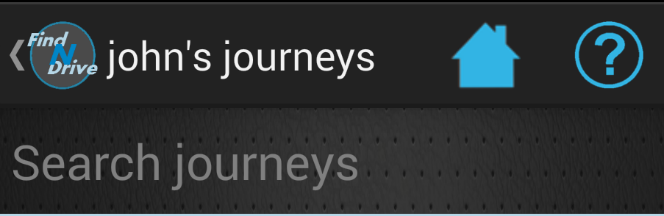


Figure Back button built into the action-bar

#### 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.

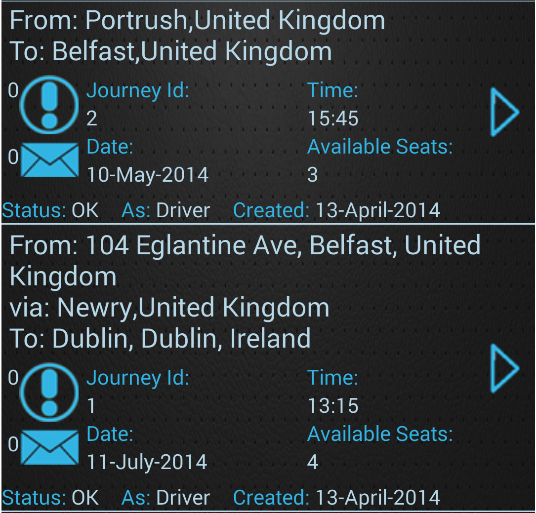


Figure ListViews are very flexible and the information to be displayed can be arranged in a variety of different ways.

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.

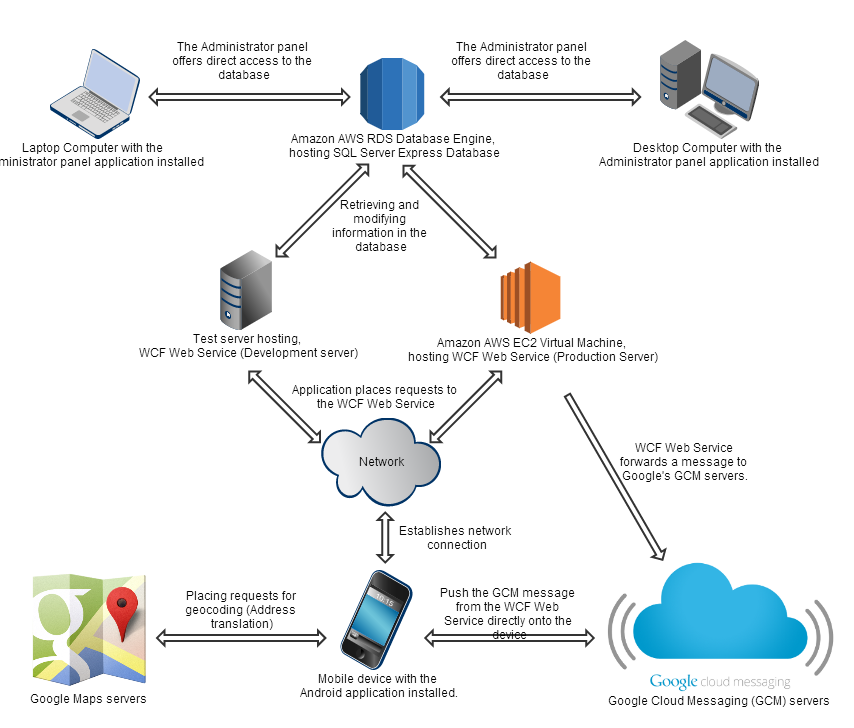


Figure . Network diagram illustrating high-level architecture of the entire system.

### 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.

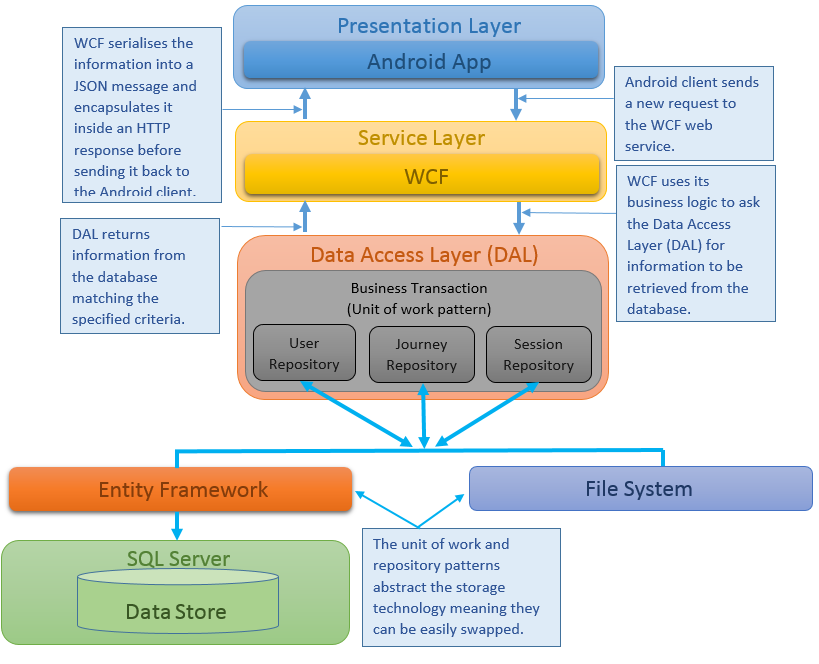


Figure illustrates the application's multi-tier design approach.

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 highest 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.

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.

#### Database Design

The application’s database will be designed using Entity Framework’s code first approach. The code first approach uses application’s domain models to generate the database. 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.

### 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.

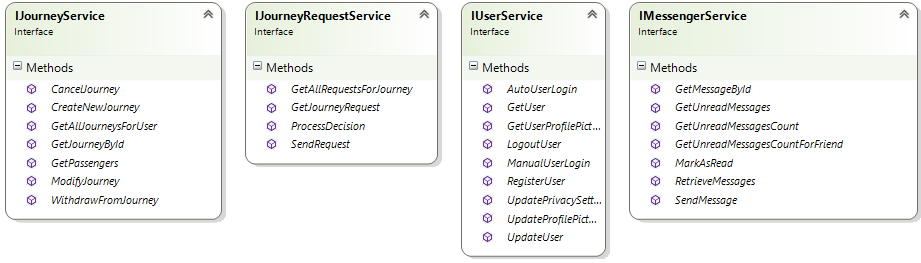


Figure 15 illustrates the interfaces which the WCF services will be implemented.

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.

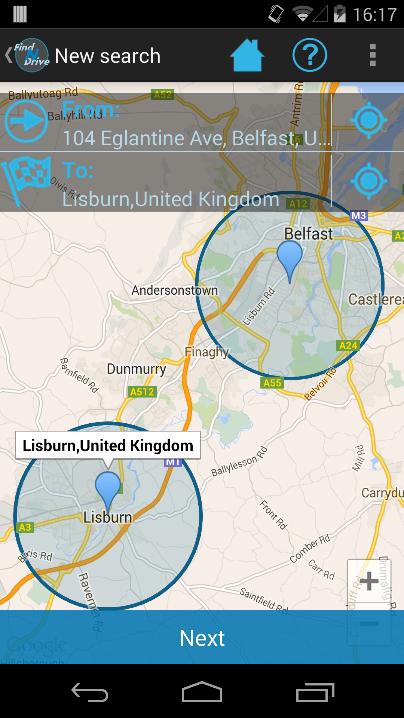


Figure . The search function making use of Google Maps to display departure and destination points of a journey that the user is searching for.

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.

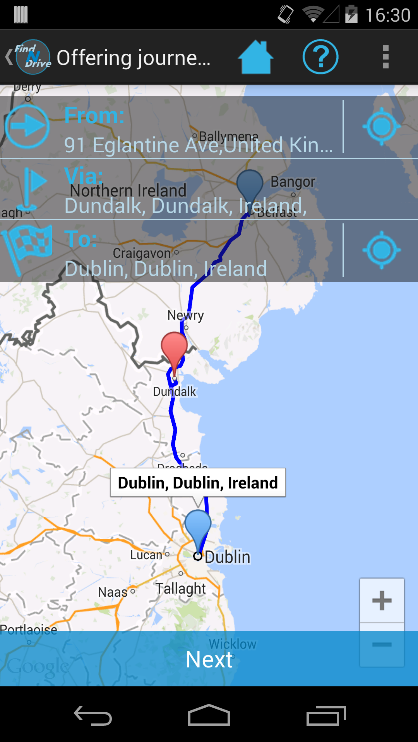


Figure . The process of offering a journey.

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.

# Implementation

This chapter focuses on the implementation of the system describing the process of transitioning the user interface and system designs from chapter 3 into the final solution.

## Development Platforms and Libraries Used.

### Mobile Application

As mentioned in section //TODO, it has been decided that the mobile application is going to target the Android platform and will be developed using Android’s native framework which uses Java as its primary development language.

The rationale behind choosing the native framework over a write-once deploy everywhere solution is the fact that the native framework produces applications capable of running directly on the device with direct access to all of the available hardware components. With native applications, there are no additional abstraction layers and no additional dependencies, meaning that native applications are more efficient when it comes to resource management which is extremely important in mobile environment where resources and battery lifespan are of a paramount concern.

Android Studio has been used as the main IDE for the Android application development. This particular IDE has been created specifically for Android development hence providing developers with many useful tools and code-refactoring features specific to Android development.

The Android application has been built with the target SDK version set to 19 meaning that the application is optimised for the latest Android 4.4 KitKat. However, to ensure compatibility with older Android versions, the minimum SDK version has been set to 11 meaning the application is compatible with Android versions as low as 3.0.

Communication between the Android application and the WCF web service takes place over the HTTPS protocol as specified in chapter 3. However, in order to ensure maximum interoperability between the Android application and the WCF web service, it has been decided that the messages exchanged by both entities will be in the JSON format. JSON being a widely recognised open-standard format, is often used to transmit data between a server and web application. With the help of JSON, it will be possible to develop additional clients using different frameworks that will still remain 100% compatible with the API exposed by the WCF web service.

Both the WCF web service and the Android application provide native support for serialisation and deserialization of JSON objects. However, in order to provide support generic objects which are described later in this chapter, Google’s GSON library has been implemented into the project.

The application also makes use of Google Play Services libraries in order to ensure it remains compatible with Google’s specification for using their services such as the Play Store, Google Cloud Messaging or Google Maps which are also described in this chapter.

### Web Service

Windows Communication Foundation 4.0 (WCF) has been chosen for the project’s server technology as mentioned earlier in chapter 3. WCF is a framework for building service-oriented applications which makes it easy to expose and consume software components using service contracts. This project utilises WCF’s RESTful features to expose service endpoints to the mobile application which can call them using the HTTP verbs such as GET, POST or PUT.

WCF has been chosen as the server technology primarily for its interoperability and extensibility. The default behaviour of the service can be changed by overriding appropriate methods. As an example, consider a scenario where additional HTTP headers can be easily injected into outgoing service responses as it shall be described later in this chapter. Extensibility was another factor which put WCF in advantage over other web service technologies. Additional services and extra functionality can be added simply by creating new service endpoints which operate just like traditional methods.

The web service has been hosted and made available to the client application via Internet Information Services (IIS) which is Microsoft’s web server. The web server has been installed on a Windows Server 2012 virtual machine hosted on Amazon’s Web Services (AWS) infrastructure using the Elastic Cloud 2 (EC2).

### Data Access Layer & Database Technology

The project’s database utilises Microsoft’s SQL Server Express 2012 technology. The database engine has been installed on the AWS Relational Database Service (RDS) infrastructure thus physically decoupling it from the web service. This approach will greatly improve the testing strategy allowing to quickly switch between the test and the production database.

The data access layer has been designed using Entity Framework 6 as described in chapter 3. Entity Framework is an Object/Relational Mapping (ORM) framework that makes it possible to work with relational data as domain-specific objects, eliminating the need for most of the data access plumbing code that developers need to write without it. Using the Entity Framework, developers issue queries using LINQ, then retrieve and manipulate data as strongly typed objects. The Entity Framework’s ORM implementation provides services like change tracking, identity resolution, lazy loading, and query translation allowing to focus on application-specific business logic rather than the data access fundamentals.

Together with the repository and unit of work patterns described in chapter 3, Entity Framework creates a flexible and reliable data access layer which abstracts the complexities of the underlying data storage technology.

### Source Control

A private GitHub repository has been created in order to keep the code repository securely backed-up.

## Development Process

### 4.2.1 Coding Approach

The development of the project has been carried out using an approach mostly based on the principles of agile development with certain elements borrowed from the Extreme Programming and Waterfall models. Elements adopted from agile development and extreme programming include the use of sprints, where new updated versions of the system were pushed to the code repository on a regular basis. Combining the elements of the waterfall model where the requirements are defined at the start of the project and are unlikely to change, together the flexibility and ability to respond to changes featured by the agile approach, allowed the development process to be approach from a brand new perspective as it shall be explained next.

Having a fixed set of requirements organised using the MoSCoW technique as described in chapter 1, with a clear deadline allowed for an initial development plan to be established as shown in figure //TODO.

Adaptation of certain principles from the agile approach has been extremely beneficial to development of the project. It allowed individual system features to be developed in small increments, thus speeding up the development process and making it possible to work on multiple features simultaneously. Using this particular approach where each feature was built in small increments also made it much easier to respond to changes in the system which had to be made due to architectural or design constraints. It would have been much more problematic and difficult to change the functionality of a fully developed function instead of one who’s only the core was implemented.

### Coding Style

A very specific coding style has been adopted from the very beginning of the development process in order to keep the code consistent and to ensure it adheres to the industry standards. As explained in the section below, the rules for writing client and server side code are not the same due to different languages – Java for mobile application, C# for the web service.

### Android Application Coding Style

Class fields.

Code comments.

### 4.2.1.2 Web Service Coding Style

Class fields

Code comments

## Android Application Implementation

This section focuses on the implementation of the Android application. It describes the various challenges faced during the development process together with the solutions which demonstrate how the problems have been solved.

### Setting up the environment

Certain amount of initial setup was required to ensure the development environment was ready and stable. Android Studio has been downloaded and installed as the primary IDE for development of the Android application. This was followed by the installation of the Android’s SDK development libraries which had to be linked to Android Studio to ensure the program was aware of the location of the libraries.

### Implementing Activities

As described in chapter 3, screens in Android are known as Activities. Almost all activities interact with the user, which means they are also responsible for creating and responding to user-interface events such as button clicks or text input.

Activities are contained within their own classes which extend one of the Android’s Activity base-classes such as the ‘Activity’ or ‘ActionbarActivity’ classes. Development of an activity takes place by overriding different methods from the base class which are called at different stages during the activity’s life cycle. At this point, it is very important to explain the activity’s life cycle understanding of which is vital to efficient Android application development.

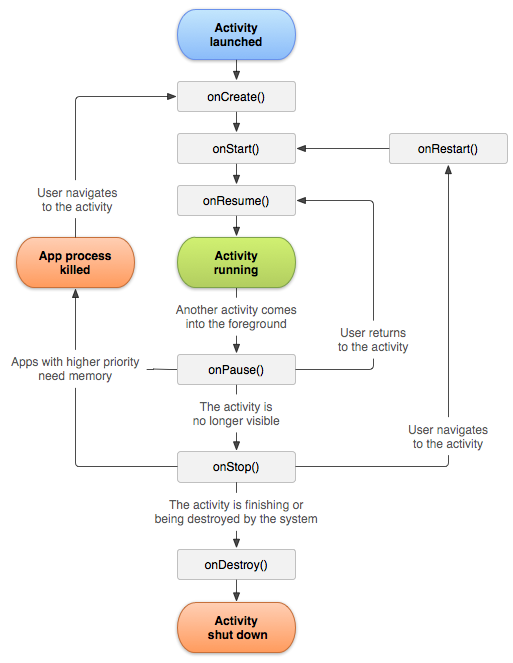


Figure Activity lifecycle.  
Source: https://developer.android.com/reference/android/app/Activity.html

Android activity lifecycle has been designed specifically with resource management in mind. As an activity is first launched, it goes through a number of initialisation stages before it’s displayed on the screen.

Closing an activity follows a similar process where an activity is first paused and kept in the stack for a period of time together with all its variables still in memory should the user return to it any time soon. If after a period of time, if the user does not return to the activity, Android destroys the activity together with all its data and variables.

Figure 19 illustrates the activity lifecycle process together with different methods called at various stages.

Each of these methods performs a very specific task. The onCreate() method as an example, is called when the activity is first created and is where the activity’s layout should be initialised and all of the variables instantiated. In order to implement own logic into activity, it is required to the relevant methods exposed by the Activity base class. The project makes use of a three-level inheritance to reduce the amount of duplicated code and to generify the tasks common to all activities such as initialisation of variables. As an example please see figure 20 illustrating the three-level inheritance.

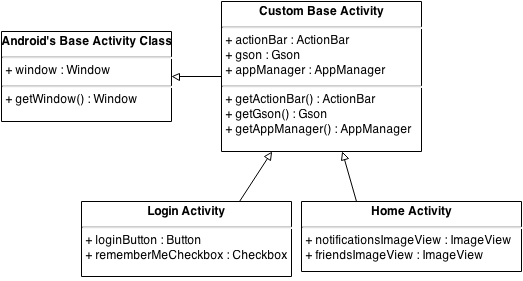


Figure Three-level activity inheritance.

Passing data between activities is achieved using Intent objects. Intents in Android, describe an operation that needs to be performed such as starting a new activity or sending broadcast. One of the main limitations of passing data between activities through intents is they only support the Framework defined types such as Strings, Integers and Booleans etc. To overcome this problem, it has been decided to use GSON which was already implemented into the project as described in section 4.3.6,

### Developing User Interfaces

Development of user interfaces has been performed with Android Studio’s built in designer using XML code to create and position individual user-interface components on the screen.

### Network connectivity & Multi-threading

Any operation that is likely take an extended period of time to complete, such as file I/O or a network request, must be performed in an asynchronous mode in order to keep the main user interface thread responsive.

Android provides its own class for performing asynchronous tasks called AsyncTask which is very similar to Java’s Thread class. In order to provide a uniform way of handling network calls, a generic BaseServiceTask class, capable of handling all types of objects has been created by extending Android’s AsyncTask class. Two additional sub-classes have been developed to extend the functionality of the BaseServiceTask, these are as follows:

**WcfPostServiceTask** is used for posting information to the server using HTTP’s POST verb. It employs Apache’s HttpPost client to send information to the server inside the HTTP request body. PostServiceTask can be used for tasks such as a registering a new user or performing a search, where a complex object with user’s search criteria is passed inside the HTTP request body.

**WcfGetServiceTask** on the other hand, is used for retrieving information from the web service using HTTP’s GET keyword. HTTP GET requests cannot have a request body and all additional parameters should added to the URL in the form of a querystring. This makes HTTP GET ideal for quick retrieval of information which can be identified quickly using as little number of parameters as possible.

If an activity wishes to perform a call to the server to either post or retrieve some information, it must implement an interface with a call-back method allowing the NetworkTask object to return a result once the call to the web server is complete. This allows the Activity to remain responsive and perform other tasks while the NetworkTask executes in the background and returns a result via the call-back method. Figure 21 illustrates the workings of the NetworkTask class where the task is started and sometime later, it returns the result via the **‘onServiceCallCompleted()’** method back to its original creator.

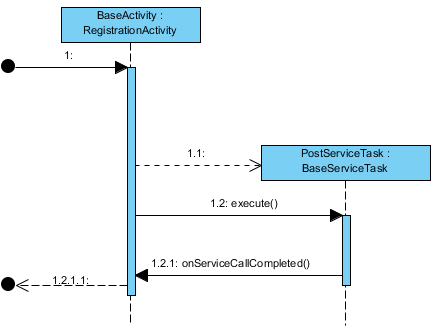


Figure PostServiceTask Sequence Diagram.

### Data Serialisation/Deserialization

One of the biggest challenges faced during the development process was creating a uniform way of deserializing objects of any type received as a response from the web service using only a single class in order to avoid code duplication.

With the help of Google’s GSON library, which provides full support for serialization and deserialization of generic objects, it was possible to extend both ServiceTask classes class described in section 4.3.4 to provide support for all types of objects to be received from the web service using a single class. This generic behaviour has been achieved with the use of GSON’s TypeToken class which tells GSON what specific type the generic object received from the web service should be converted to.

Figure 22 illustrates the process of contacting the web service to retrieve the list of user’s journeys. When a new PostServiceTask is created, it receives an instance of the TypeToken class as one of its constructor parameters. In this particular case, the TypeToken tells GSON the object it’s about to receive for deserialization is of type ArrayList<Journey> since the web service call was made to retrieve all of the user’s journeys. GSON then uses this TypeToken object to parse the response it receives from the web service into the ArrayList<Journey> type which is then passed back to the Activity via the call-back method.

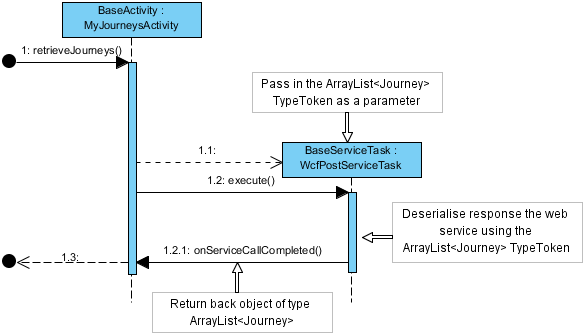


Figure 22 the process of deserialising generic object into a specific type with the use of Type Tokens.

### Persisting data

Due to the nature of Android’s resource management system, any data required by the application to run properly should be persisted onto the device’s storage at regular intervals. This is because Android can erase the application’s data from memory if it’s running out of resources and the application has been inactive for some time. This means that all of the information that was previously kept in memory will be gone when the user returns to the application at a later stage.

To solve this problem, an application manager has been developed in order to persist the data required by the application to run properly and then reload it from device’s storage if the application’s memory has been erased.

The application manager provides global access to some of the application’s most important data, such as the currently logged in user, or the current session information, required for session authentication on the server side.

Behind the scenes, the application manager uses Android’s SharedPreferences class to persist data as its values are being assigned. SharedPreferences uses the device’s storage to persist data in the XML format. On the other hand, when data is retrieved from the application manager, it first checks if the object is still in memory by if its reference is null. If the reference is null, it means that the variable has been cleared from memory by the operating system and must be retrieved from device’s storage before it’s returned.

### Client session management

Session management on the client side has been restricted to a small set of simple operations, these include:

* Persisting session information received from the server after successful login using SharedPreferences described in section 4.3.7.
* Injecting the persisted session information into HTTP headers when making web service calls to allow the WCF web service to perform authentication of the user.

### Google Maps, GPS, Geocoding.

As mentioned in chapter three, the application relies heavily on Google Maps to perform address geocoding and to display journey route on the map. One of the requirements for implementing Google Maps into the project was obtaining a valid Google Maps API key via Google’s APIS console. This involved registering the project with Google and enabling the Google Maps API for the project to make use of.

Once the key was obtained and installed into the project, Google Maps became fully accessible and could be used within the application. Plotting points on the map has been achieved with markers. Markers are objects which contain all the information necessary to identify point on a map, such as latitude and longitude, address or even a bitmap image which visually represents the marker.

Geocoding an address and showing it on the map is a process which must be performed in its own thread due to its time-consuming nature as described in section 4.3.5. For this reason, a GeocoderTask class has been implemented the sole purpose of which is perform the geocoding process on the address provided and return back the result as soon as the process completes.

One of the challenges faced when using the Android’s Geocoder class was the fact that it is only available on devices that are fully compatible with Google Play services. The solution involved developing a backup Geocoder class which is to be used when the primary Android Geocoder is not available. The backup Geocoder uses Google’s geocoding web API to perform the geocoding process and parse the JSON response containing the address information. Figure 23 illustrates the process of choosing between the primary and secondary geocoders depending on the availability of the primary one.



Figure falling back on the backup geocoder if the primary one is unavailable.

In order to provide for a better user experience, the application implements GPS functionality to allow the user quickly get a hold of their current location either when advertising or when searching for journeys.

The Android framework provides an easy access to the device’s GPS functions meaning that implementing GPS functionality into the application was a relatively straightforward process. However, one very important factor that had to be considered was the level of accuracy and the frequency of location updates that are required by the application. Higher levels of accuracy and higher frequencies of location updates have a very negative effect on the device’s battery lifespan. It was therefore extremely important to find optimal balance using appropriate code, to ensure the application had as little effect on the battery lifespan as possible.

There are a number of strategies available to developers when it comes to retrieving current location. The one that has been implemented into the application involves using the last remembered location retrieved from the cache. This approach provides current location that’s sufficiently accurate while having minimal to no effect on the battery life. Should there be no location stored on the device’s cache, the application will request the device to start the location acquiring process and return the result back as soon as it’s available.

### Working with Images and LRU image cache.

Almost every Android device on the market has a hardware camera device. The application makes use of the device’s camera to provide users with the ability to change their profile picture.

In order to make use of the camera, the application must launch a special activity whose intent is set to ACTION\_IMAGE\_CAPTURE. Once the activity is started, it opens the device’s default camera application which allows the user to take a picture and return it back to the original activity for further processing.

This further processing involves resizing and compressing the image to reduce its size and optimize of for transfer across the network. The BitmapUtilities class has been created to expose static helper methods that perform the image resizing and return a much smaller and compressed bitmap image. The code for the BitmapUtilities class is located on page //TODO in the appendices.

Caching images retrieved from the web service is an extremely important process. Ideally, an image should only be retrieved once and stored in cache, from which it should be loaded the next time it is required. The application utilises Android’s Least Recently Used (LRU) in-memory cache to store images in the memory after they have been downloaded for the first time.

After an image is stored in cache, all subsequent requests for the image are loaded from the LRU cache greatly speeding up the loading process and improving the device’s battery lifespan due to not having to perform another network call. The LRU cache utilises a hash-map structure allowing items to be stored using a {“key”, “value”} system where the key is a unique identifier of the item being stored as the value. This greatly speeds up the process of checking if an image is already present in the cache since the only check that is required is to see if the key’s value is null. The below steps illustrate the mechanism of loading an image for the first time.

* The application is asked to load an image into an ImageView control.
* The LRU in-memory cache is checked to see if the image has already been downloaded.
* The image was not found in the cache, the application calls the WCF service to retrieve it.
* The image is downloaded and stored in the cache using the {“key”, “value”} system.
* All subsequent requests for this image are retrieved from the cache using the key.

### Input Validation

Input validation on the client side has been handled by using custom validation functions. Those functions have been declared as static to make them accessible globally to all other functions. Below is a list of validation functions that have been defined.

* validateEmailAddress – this function uses a regex pattern to detect whether the string entered by the user is of a valid email address format. It is used during new user registration and inside the profile editor activity which allows the user to change their email address. The regex expression used to validate the email address is as follows:   
  *"^[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+(?:\\.[a-z0-9!#$%&'\*+/=?^\_`{|}~-]+)\*@(?:[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?\\.)+[a-z0-9](?:[a-z0-9-]\*[a-z0-9])?$"*
* validatePasswords – this function is used during new user registration to ensure that the two passwords supplied by the user are at least six characters long and also to check that both passwords match.
* validateUserName – this function validates the username during new user registration to ensure it is at least six characters long and that it does not contain any illegal characters.

If an error is found during one of the data validation checks, the current task is immediately cancelled and the user is notified about the erroneous piece of information they have entered. Android provides a setError method on the TextView and EditText user-interface controls to inform the user of an error together with an error message. Figure 24 below shows the error message being displayed inside an EditText control.

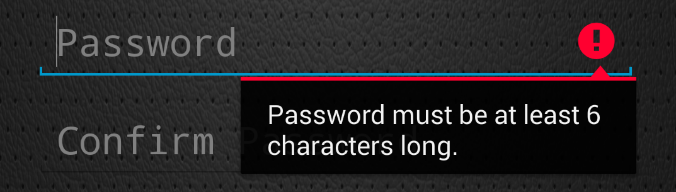


Figure Android's setError message informing user of a data validation error.

### Sending Broadcasts

## WCF Web Service Implementation

### Setting up the environment.

Development of the WCF web service took place in Microsoft Visual Studio 2012 Ultimate. The initial setup involved installation of the SQL Server Express instance for the test development server as well as Amazon’s AWS Visual Studio plugin to enable quick deployment of the WCF services to Amazon’s EC2 virtual machine which is the designated production server.

### Implementing Service & Operation Contracts

WCF services expose various operations through methods marked with the ‘OperationContract’ attribute. These methods are declared within an interface marked with the ‘ServiceContract’ attribute. Each of the services defined in the project has an accompanying interface whose methods it implements to add extra logic.

### Implementing Data Access Layer

As described in chapter 3, the Data Access Layer has been implemented using Entity Framework together with the generic repository and unit of work patterns. It forms an abstraction layer hiding the complexities of the underlying data storage technology allowing for development in the form of domain-specific objects and properties. This section provides a more detailed description of the implementation process of the Data Access Layer.

The implementation process consisted of three main stages:

* Designing & implementing application data models.
* Implementing the generic repository and unit of work patterns.
* Generating database using the code first approach.

#### Designing & implementing application data models

Designing and implementing the data models was a continuous process which required constant updates and a lot of attention. As new features were developed, existing data models evolved and new models were created. All of the data models have been encapsulated within their own classes. The values of their fields are retrieved and set through public properties. The initial set of data models were basic and have served as a foundation for development of the more complex ones.

As mentioned in chapter 3, Entity Framework 6.0 has been chosen as the primary technology for the Data Access Layer. Entity Framework is an open source object-relational mapping (ORM) framework for ADO.NET, which is a library that exposes data access services for the .NET framework.

Entity Framework uses the data models such as the User Model or the Journey Model to create mappings between the database schema and the conceptual schema. It uses the model’s fields such as UserName or DateOfBirth in the case of the User Model, and maps them to physical tables in the database.

#### Implementing the generic repository and unit of work patterns.

The repository and unit of work patterns are essentially at the core of the abstraction layer. The repository pattern provides all the necessary CRUD operations through a set of data manipulation methods which communicate with the data source. These methods are as follows:

* **public void Add(T entity)** – adds a new entity of type T to the repository.
* **public void Remove(T entity)** – removes an existing entity of type T from the repository.
* **public void RemoveRange(IEnumerable<T> entity)** – removes an entire collection of entities of type T from the repository.
* **public T Find(int id)** – retrieves a specific entity by its unique identifier.
* **public IQueryable<T> AsQueryable()** – allows to retrieve a list of entities using LINQ and Lambda extensions available in C#.

Implementing a generic repository class allowed a single repository class to be used with all domain model types thus eliminating a lot of redundant code. Each of the domain models have been placed into their own repository through which they will be accessible using the above set of methods. As an example, consider the code snippet in figure 26 which is used to retrieve the use whose user ID is equal to 2.



Figure retrieving the user whose user ID is equal to 2 using the UserRepository.

Another example in figure 27 shows how the LINQ and to entities together with lambda expressions can be used to easily create a more complex query without the need to write complicated SQL code. In this case, the query retrieves a list of users whose first name is Tom and whose rating is bigger than 4.



Figure creating a more complex query using LINQ and Lambda expressions.

The unit of work pattern on the other hand contains a list of all of the repositories present within the system and uses them to, handle transactions during data manipulation. All the repositories share a single database context meaning that all the changes made to entities which reside in different repositories can be committed to the database using a single transaction. The ‘Commit()’ method which the unit of work exposes coordinates the process of saving the changes to the database in a single transaction also taking care of the concurrency problem. The code snippet in figure 28 illustrates the process of updating the email address whose user ID is equal to 2. The process is as follows:

* Retrieve the user with the user ID 2 and keep a reference to it in the ‘user’ variable.
* Update the user’s email address by changing the value of the EmailAddress property.
* Write the changes back to the database using the ‘Commit()’ function exposed by the unit of work.



Figure using the unit of work pattern to update user's email address.

#### Generating database using the code first approach.

Entity Framework’s code first approach allowed for a rapid database generation using a single ‘update-database’ command. As mentioned in chapter 3, the code-first approach allows a more code-centric approach where the database schema is generated from the classes which represent the domain models. One of the biggest challenges faced when using the code-first approach was defining the relationships between the entities. Entity Framework allows for the relationships between entities to be defined either by using data annotations or c# statements known as fluent API. The former has been implemented into the project the evidence of which can be seen in figure 28. The figure illustrates the fluent API approach used to define one-to-many relationship between the journey object and its passengers. The left part of the image shows the fluent API syntax used to define the relationship while the right side of the image shows the resulting mapping table in the database used by Entity Framework to keep track of which users belong to which journeys.

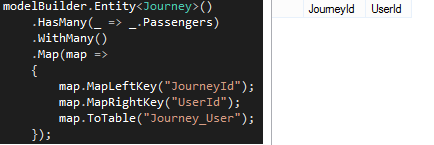


Figure fluent API used to define one-to-many relationship between the journey and its passengers.

### Location-aware search processor

As described in chapters two and three, the idea behind the location-aware search is to provide users with journey recommendations. As an example, consider the following scenario where the user searches for a journey whose departure point is in City A street A1. The search engine scans through the list of journeys currently present in the database and does not find a journey whose departure point is an exact match for user’s criteria. However, the user also specified that they will consider all journeys whose departure points are within a 2 mile radius of city A street A1. The search engine sees that one of the journeys departs from city A street A2 which is within the 2 mile radius and therefore should be returned in the search results.

Implementing this functionality was possible with the help of the Haversine formula, which is demonstrated in figure 29. Haversine formula is an equation used in navigation giving great-circle distances between two points on a sphere from the latitude and longitude values. In other words, this equation is used to calculate the distance between two points on a map taking the curvature of the earth into consideration to provide extremely accurate results.



Figure the Haversine formula.

The C# code implementation of the Haversine formula is shown in figure 31.

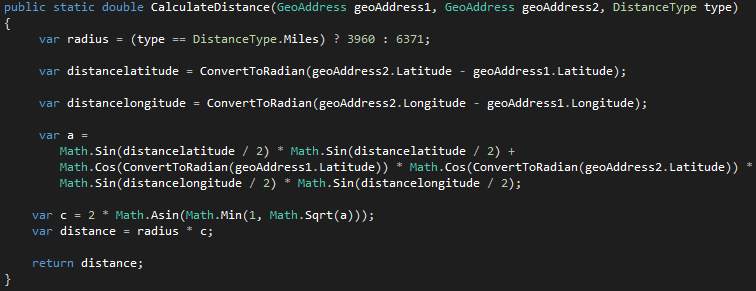


Figure code implementation of the Haversine formula.

Another important feature offered by the search processor is the ability to identify that a journey being searched for might be a smaller subset of a bigger journey that already exists in the database. As an example, consider that a journey object exists in the database with the following destinations:   
  
 { Journey A: { GeoAddresses: City A -> City B -> City C -> City D -> City E } }

Now let us assume that the user searches for a journey with the following destinations:

{ Journey B: { GeoAddresses: City A -> City C} }

As it can be seen, the smaller journey object happens to be a smaller subset of the bigger journey. The search processor would therefore consider the bigger journey to be a valid match for user’s search criteria because the cities and the order in which they’re being visited matches, that is City A is visited before City C in both cases.

Identifying whether a smaller journey object is a subset of a bigger one is possible due to the system maintaining an order, in which the journey destinations are visited. When a journey is advertised, each of the destinations specified by the user is assigned an index number representing its order in the journey’s destination list. The search engine then considers this index while attempting to identify smaller subsets in a bigger journey. This prevents erroneous search results from being returned back to the user. An example of such error would be the system identifying a smaller journey to be a subset of a bigger one with the destinations in the two journeys being visited in different orders.

Other functions offered by the search processor include validation of the following search criteria:

* The number of available seats
* Whether smokers are allowed
* Whether pets are allowed
* The vehicle type specified by the user
* The maximum fee which the user wishes to pay
* User’s date and time flexibility, allowing the user to specify their flexibility in days and hours from their desired journey departure date & time. All journeys whose departure dates are within the flexible period will be considered valid for this user.

The code for the search function is located in the appendices on page //TODO.

### Deploying Services to AWS and hosting in IIS.

Services that were developed locally had to be published to the test and production servers in order to become available for the Android application. Deployment to the local test server has been achieved with Visual Studio’s one-click publish feature which deployed all of the WCF services to the locally-running instance of the IIS server.

Deploying the WCF services to the AWS Elastic Cloud 2 (EC2) instance which was the project’s dedicated production server was achieved with Amazon’s Visual Studio AWS plugin. This plugin allowed the services to be deployed using just a few clicks by automatically contacting the EC2 virtual machine and uploading the WCF services to it.

Hosting the WCF services on Amazon’s cloud infrastructure offers a number of advantages compared to hosting the services locally. Perhaps the biggest advantage is the possibility of migrating the services from the current EC2 instance to either a smaller or a bigger one with higher amount of memory and more processing power. This makes for a more efficient resource-utilisation since the services utilise all of the resources available to them and can be always be migrated to a larger instance should the demand grow.

Another advantage is the reliability and redundancy provided by the Amazon’s infrastructure. One of the big dangers of hosting the services locally is the possibility of the local server’s Internet connection suffering from a breakdown. This means all of the services will suddenly become unavailable and the Android application will have no server to contact. Using Amazon’s infrastructure, the likelihood of a similar event happening is almost non-existent since Amazon’s infrastructure consist of multiple connections to the same host having the ability to fail-over on the backup connections should the primary ones fail.

## Cross-Platform components

This section focuses on the system components whose implementation spans across both the client and the server.

### Session Management

The session management system has been developed from the grounds up for the purpose of this project. Its main goal is protection and authorization of requests received by the WCF service. Some of the functions exposed by the WCF service that modify and add new information to the database are highly sensitive. Those functions require an extra level of protection to prevent unauthorised access. The session management system provides a means of authorizing the incoming request and returning an ‘Unauthorised’ response should any of the session information mismatch provided in the request mismatch. The system is based on a set of special device identifiers that uniquely identify each user. The session management system features the following types of sessions:

* **Temporary** – this type of session is generated when the user logs into the application without ticking the ‘Keep me logged in’ checkbox. This session remains valid for 30 minutes after the initial login and is refreshed each time a request to an authorised WCF function is made. If the user does not interact with the app for more than 30 minutes, they are automatically logged out. Also if the user exists the app through the back button or the logout menu option, they are automatically logged out and will be required to login again using their username and password.
* **Permanent** – this type of session is established when the user logs into the application while selecting the ‘Keep me logged in’ checkbox. This session is set two last for two weeks allowing the user to exit the application through the back button without the need to re-login again next time they launch the application. All subsequent application launches use the session information stored on the device to perform automatic-login. In order to invalidate the session, user can use the logout option from the application’s menu.

One of the major challenges faced during the development of the session management system was the fact that Android allows the user to kill the application’s process using the built-in task manager. This posed a potential issue when a temporary session has been established between the application and the server. Killing the application’s process does not give it the chance to send a logout signal to the server hence not being able to invalidate the current session.

This allowed the user to go through the automatic log-in process despite only having a temporary session established. To overcome this problem, additional variable has been introduced. This variable holds a random string value generated at the start of the application. This value is sent to the server along with the login request. If this value is found to be different from the one that has been stored in the database during the initial login request before the application was killed, a failure is returned, the current session is immediately invalidated and the user is asked to log in again.

An advantage of using this approach is the fact that user can only be logged in on one device simultaneously. This is because as soon as the user logs-in from a second device without first logging out from the first one, the session information for this user will be immediately overwritten with the details of the second device thus invalidating the session between the first device and the server.

Figure 25 illustrates a high-level overview of the session management system providing a description for each of the various session validation stages.

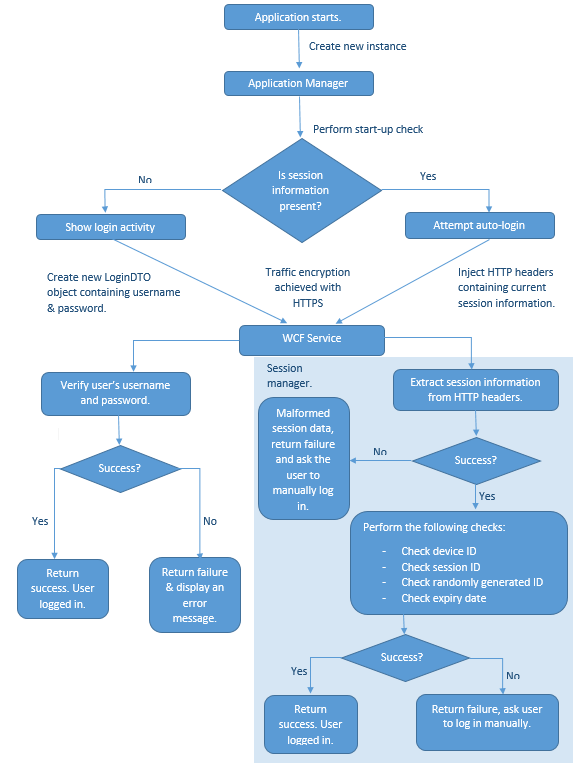


Figure Overview of the session management system.

### Notification System

As mentioned in chapter 1, the notification system has been designed to fully utilize the notification functions offered by the Android platform. The notification system serves as a backbone for all of the application’s social features and the built-in chat functionality. This section focuses on the implementation of the notification system on the client and the server. The notification system has been designed to provide a way of informing the user immediately of an important event by sending the notification directly to their device.

#### Notification Types

The notification system features two main types of notifications which users can interact with:

* **In-App** – this type of notification can be accessed through the Android application’s ‘Notifications’ menu. These notifications inform the user of an event that does not require their immediate attention and can be viewed in user’s own time. In-app notifications are not intrusive and the event of a new in-app notification arriving is signified by the ‘Notifications’ icon changing colour to grey.
* **Device** – this type of notification can be considered slightly more intrusive as it will wake the device from sleep if required. Device notifications appear in the device’s top-left corner on arrival immediately alerting the user of an event.

The major components of a notification object include:

* **CollapsibleKey (Integer)** – used to prevent multiple notifications with the same intent from being displayed on the device. For example, when a new message from one of user’s friends is received, a notification is displayed. However, if the same friend sends another message, the **CollapsibleKey** is used to identify the notification that’s currently being displayed to update it with the new message instead of displaying both notifications.
* **NotificationContentType (Enum) –** used to identify the content type of a notification.
* **NotificationMessage (String)** – contains the textual message that the user can read when the notification is displayed.
* **NotificationPicture (Integer)** – Identifies the id of the notification picture which needs to be retrieved from the database before the notification is displayed.
* **TargetObjectId (Integer)** – identifies the id of the object which needs to be retrieved from the database before the notification is displayed.

#### Downloading & displaying notifications

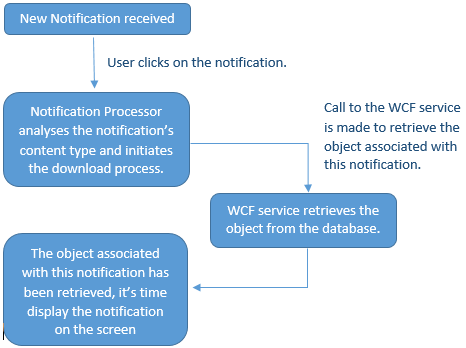
A notification may have another object associated with it depending on its content type. For example, a notification informing the user of a new friend request whose content type is set to   
**‘NotificationContentType.FriendRequestReceived’**, will have a FriendRequest object associated with it.

This allows the notification processor implemented in the Android application to use this FriendRequest object to launch the relevant dialog window allowing the user to make a decision regarding this friend request after the notification is clicked.

Rather than including the object in the notification itself, the notification keeps a reference to the ID number of the target object that must to be retrieved from the database before it’s displayed on the screen. This is because of the message size limit imposed by Google Cloud Messaging making it impossible to include larger objects inside a notification.

The process of downloading an object associated with a notification and displaying it on the screen is illustrated in figure 32.

Figure retrieving notification object and displaying it on the screen.



#### Notification Manager

The notification manager is at the heart of the system’s notification infrastructure. The notification manager is responsible for creating and sending notifications directly to the user’s device with the help of Google Cloud Messaging (GCM).

The notification manager exposes the following methods:

* **SendGcmTickle** – this method sends a notification tickle to all of the users in the collection which is passed into the method. Notification tickle acts as a signal for the device letting it know that there is new data waiting for it on the server and that it must synchronise its state. Sending a notification tickle and asking the device to download new data is preferred to including the actual data in the notification itself. This is due to the fact that GCM imposes limitations on the size of the message meaning it would be impossible to send larger objects inside a notification.
* **SendMessage**– this method is used by majority of the services in the system to send device notifications. A service wishing to send a notification to the user informing them of an event such as their friend advertising a new journey can use this method, to send an immediate notification to all of the user’s friends who will be able to click on the notification and view the journey advertised by the user.
* **CreateAppNotification**– this method creates a new notification and adds it to the Notifications repository. Calls to this method are always followed by a call to the **SendGcmTickle** method to send a synchronise signal to the Android application which will download the notification which has just been added to the repository.
* **ForwardGCMNotification –** this method uses the prepared JSON statement compatible with Google Cloud Messaging message format to forward the notification onto Google’s server which in turn push the message directly onto user’s device. Figure 33 illustrates the JSON statements used for sending notifications onto GCM servers.

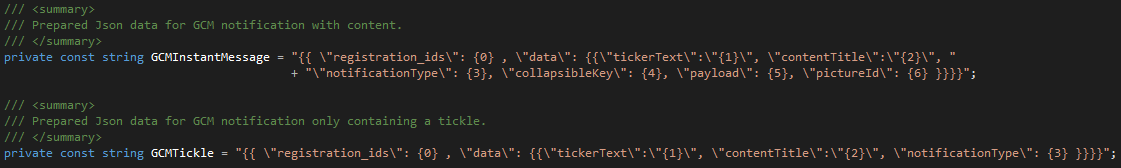


Figure Prepared JSON statements used for pushing notifications onto GCM servers.

Another important function of the Notification Manager is the ability to distinguish between which users are currently online and which ones are offline. The notification manager loops through the collection of users which it receives as one of the parameters and if any of the users are found to be offline, the notification is saved in the NotificationRepository which will be downloaded by the device as soon as they log in. Users who are found to be online at the time of iterating through the collection will have the notification dispatched directly to their devices immediately.

#### Multi-user chat

Multi-user chat is an extended version of the one-to-one instant-messenger feature built into the system. It allows message to be exchanged in real-time between all the passengers of a particular journey. Messages are broadcast to all of the passengers of a journey. The notification manager analyses the list of passengers to check their current online status. Any passenger whose current online status is offline will receive the message immediately after logging-in, in the meantime, the message will be saved in the NotificationsRepository by the NotificationManager. Figure 34 illustrates the functionality of the multi-user chat system.

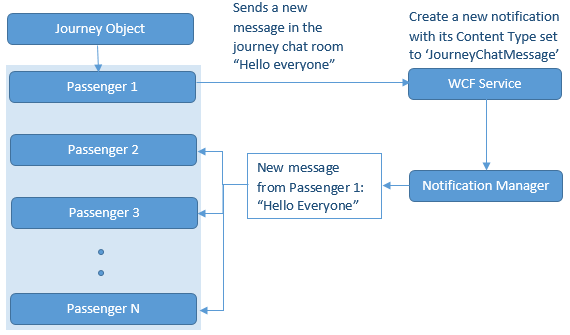


Figure High-level overview of the multi-user chat architecture.

## Test Strategy and Test Suite

The test strategy which has been adopted for the purposes of this project has proven very successful. It employs a variety of different types of tests including user-interface testing, unit testing, acceptance/integration testing and performance testing measuring the performance of the system under stress.

### Unit Testing

Unit tests have been written on both the client and the server side to test the functionality of individual the components independent of all other components within the system.

//TODO – insert appendix reference showing test results.

#### Android Application Unit Testing

Android’s native InstrumentationTestCase class was utilised on the client side to write and run the unit tests.

#### 4.6.2.2 Server-side Unit Testing

Unit tests on the server-side have been utilised to test the functionality of components such as the Search Processor, the Notification Manager, the Session Manager, the Haversine Calculator or the Email Utilities.

### User-Interface Testing

Android’s test framework enables the creation of automated user-interface tests which interact with the various user-interface elements on the screen in exactly the same way a human user does. Using Java code, it’s possible to simulate button presses, text input and much more. This provides a fantastic way of quickly testing the user-interfaces in code without having to manually go through the various activities and pay attention to how various user-interface elements react to input.

//TODO – insert appendix reference showing test results.

### Integration/Acceptance Testing

The integration and acceptance tests are the most the important group of tests within the project for one reason: they test the entire application stack, starting at the business layer, testing the functionality of all of the components all the way down to the data access layer. Those tests utilise real HTTP web requests and call real WCF service on the test server which uses a real SQL server test database. For example, the integration tests for the Friends Service test the functionality of the Friends Service itself by sending and responding to a friend request in code, as well as the Notification Manager, Session Manager and Email Utilities all of which are used inside the Friends Service.

//TODO – insert appendix reference showing test results.

### Performance Testing

One of the new features recently implemented into Visual Studio is the ability to carry out performance testing. Performance testing is a way of measuring the system’s ability to cope under stress. It is done by simulating a pre-defined number of virtual-users who hit the system with a variety of requests simultaneously.

The performance tests in this project have been configured to use the integration tests described in section 4.6.3 behind the scenes, as a means of allowing the virtual-users to interact with the service. The tests have been configured to use different proportions of various services as well as to call different functions and different times. The tests can be configured to last a pre-defined length of time or can run through a pre-defined number of iterations. For the purposes of this project, the tests have been configured to run for a period of 2 minutes. While the tests are running, Visual Studio records all the important statistics such as the number of failures, average response time and much more. For more detailed statistics and the actual test results, please refer to page //TODO, in the appendix.

# 5 Conclusion

# Appendix

**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 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. |