*Acknowledgments*

*Abstract*

(Quick summary of the dissertation)

# Introduction.

The aim of the project is to create an application for mobile devices designed for car sharing. Using their handheld devices, users of the application will be able to advertise their own 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 served as the foundation for my problem specification:

**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.

To measure the usability of the application, a benchmark against other car-sharing services has been established. The below list of features represents the core functionality offered by all of the car-sharing services currently in existence. It’s important to note however, that the below list represents an absolute-minimum 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 however, 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. **(Modified)**
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 list of changes that have been made to the original list of requirements.

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.

18. 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. //TODO explain why Android? Not iOS or Windows Phone?
* **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. //TODO, Explain RPC with RESTful subsystem?

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. It must be noted however, that the models specified in this section are not the exact representation of the application’s domain objects. The below data models should be considered as a guideline when implementing the data models in code and their structure may differ.

## Data Model

### 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.

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.

**//TODO – Insert diagram illustrating how DTO’s benefit the application.**

The second group of the data model consists of domain objects which are mapped directly onto tables inside the database. With the help of Entity Framework, the generic repository and the unit of work patterns, the business logic inside the web service uses applications’ domain objects to manipulate information stored in the database directly. For more information on the implementation details of Entity Framework please go to chapter 4.

**//TODO – give exact reference to chapter 4 when it’s done.**

The section below demonstrates the structure of applications’ domain objects that have been derived from the requirements and are part of the application’s data model.

### Unmapped fields.

In order to further reduce the number of calls made by the client and to avoid the circular dependencies problem present in some domain objects, the concept of unmapped fields has been introduced. For a detailed explanation of the circular dependency problem and its solution please refer to chapter 4.

**//TODO – Insert the reference to circular dependency problem in chapter 4 when done.**

### User object.

Represents a real-world user entity that registers and interacts 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 .. Int Max. | Uniquely identifies each user in the database. |
| ProfilePicture | ProfilePicture | N/A | Contains user’s profile picture. Please see the ProfilePicture object for more details. |
| UserName | String | 6 .. 30 char. | Uniquely identifies user in the system. |
| EmailAddress | String | 6 .. 50 char. | Contains user’s email address. |
| First Name | String |  | Contains user’s first name. |
| LastName | String |  | Contains user’s last name. |
| DateOfBirth | DateTime |  | Contains user’s date of birth. |
| Gender | Enum |  | Identifies user’s gender. |
| GCMRegistrationID | String |  | Contains user’s GCM Registration ID assigned to user’s device by Google’s GCM servers. |
| Friends | List<User> |  | Contains a list of User objects identified as user’s friends. |
| Status | Enum |  | Contains user’s current online status. |
| Ratings | List<Rating> |  | Contains a list of Rating objects representing this user’s ratings left by other users. |
| LastLogon | DateTime |  | Contains the date and time of this user’s last logon. |
| PhoneNumber | String |  | Contains user’s phone number. |
| AverageRating | Double |  | Contains user’s current average rating. |
| JourneyTemplates | List<JourneyTemplate> |  | Contains a list of JourneyTemplate objects each of which represents a JourneyTemplate created by this user. |
| Notifications | List<Notification> |  | Contains a list of Notification objects each of which represents a different notification targeted for this user. |
| PrivacySettings | PrivacySettings |  | Contains the user’s privacy settings which control which aspects of their profile are visible to other users and friends. |

### User object – unmapped fields.

|  |  |  |
| --- | --- | --- |
| UnreadMessagesCount | Integer | Contains the number of unread messages from this user when user’s friends list is loaded. |
| JourneysVisible | Boolean | Identifies whether user’s journeys are publicly visible. The value of this variable is based on the user’s privacy settings. |
| RatingsCount | Integer | Contains the total number of ratings that this user has received. This variable is displayed next to user’s name in the global leader board. The actual list of ratings is retrieved from the web service using a separate call. |

### Journey object.

Represents a journey object that users can create and send requests for.

|  |  |  |
| --- | --- | --- |
| **Name** | **Data Type** | **Description** |
| JourneyId | Integer | Uniquely identifies each journey object in the database. |
| Driver | **User** | **Identifies the user who offered this journey.** |
| **GeoAddresses** | **List<GeoAddress>** | **A collection of GeoAddress objects which contain addresses and locations of this journey’s start and end locations as well as any optional waypoints.** |
| **DateAndTimeOfDeparture** | **DateTime** | **Identifies the departure date and time of this journey.** |
| **Description** | **String** | **Contains a short message created by the driver visible to everyone who wishes to apply for this journey.** |
| **Fee** | **Double** | **Contains the fee for this journey specified by the driver.** |
| **AvailableSeats** | **Integer** | **Contains the number of available seats specified by the driver.** |
| **Passengers** | **List<User>** | **Collection of User objects containing all the users who have applied for this journey and have been accepted by the driver.** |
| **Pets** | **Boolean** | **Specifies whether Pets are allowed in this journey.** |
| **Smokers** | **Boolean** | **Specifies whether Smokers are allowed in this journey.** |
| **Private** | **Boolean** | **Specifies whether this journey is only visible to Users from the driver’s friends list.** |
| **VehicleType** | **Enum** | **Contains the vehicle type specified by the user during journey creation.** |
| **UndecidedRequestsCount** | **Integer** | **Contains total number of requests submitted by other users where no decision was made by the driver.** |
| **JourneyStatus** | **Enum** | **Contains the current status of this journey.** |
| **JourneyRequests** | **List <JourneyRequest>** | **Collection of JourneyRequest objects that have been created for this journey.** |
| **CreationDate** | **DateTime** | **Contains the date and time when this journey was created.** |
| **PreferredPaymentMethod** | **String** | **Contains the preferred payment method for this journey as indicated by the driver.** |
| **UnreadMessagesCount** | **Integer** | **Contains the total number of unread messages for the currently logged in user for this journey.** |

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

* JourneyRequestId (int)
* JourneyId (int)
* Journey (Journey)
* FromUser (User)
* Message (string)
* Read (Boolean)
* Decision (Enum)
* SentOnDate (DateTime)
* DecidedOnDate (DateTime)

**FriendRequest** - 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.

* FriendRequestId (int)
* FromUser (User)
* ToUser (User)
* Message (string)
* Decision (enum)
* Read (Boolean)
* SentOnDate (DateTime)
* DecidedOnDate (DateTime)

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

* UserId (int)
* DecideId (string)
* SessionId (String)
* ExpiryDate (DateTime)
* SessionType (enum)

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

* RatingId (int)
* UserId (int)
* TargetUser (User)
* FromUser (User)
* LeftOnDate (DateTime)
* Score (int)
* Feedback (string)

**ProfilePicture** – 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 a number of design decisions:

* Profile picture only needs to be downloaded once and stored in Least Recently Used (LRU) memory cache, there is no need to retrieve the picture from the server each time the user object is requested.
* Retrieving the user object without the profile picture results in a much smaller HTTP response thus speeding up the object serialisation and deserialization on both, the web service and the mobile application.

Profile Picture

* ProfilePictureId (int)
* ProfilePictureBytes (byte[])

**PrivacySettings**  - 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.

* PrivacySettingsId (int)
* EmailPrivacyLevel (enum)
* GenderPrivacyLevel (enum)
* DateOfBirthPrivacyLevel (enum)
* PhoneNumberPrivacyLevel (enum)
* RatingPrivacyLevel (enum)
* JourneysPrivacyLevel (enum)

**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.

* NotificationId (int)
* UserId (int)
* User (User)
* ProfilePictureId (int)
* CollapsibleKey (int)
* NotificationType (enum)
* NotificationContentType (enum)
* NotificationMessage (string)
* TargetObjectId (int)
* Delivered (Boolean)
* ReceivedOnDate (DateTime)

**JourneyTemplate** – represents a journey template object used to enable automatic notifications and save users from the effort of performing manual search. Explain why duplicate data exists here.

* JourneyTemplateId (int)
* Alias (string)
* UserId (int)
* User (User)
* Fee (double)
* DepartureRadius (double)
* DestinationRadius (double)
* Pets (Boolean)
* Smokers (Boolean)
* VehicleType (Enum)
* GeoAddresses (List<GeoAddress>)
* DateAllowance (int)
* TimeAllowance (int)
* DateAndTimeOfDeparture (DateTime)
* SearchByDate (Boolean)
* SearchByTime (Boolean)
* CreationDate (DateTime)

**JourneyMessage** – represents a journey message object, used in multi-user journey chat rooms to transfer the messages.

* JourneyMessageId (int)
* JourneyId (int)
* SenderId (int)
* SenderUserName (string)
* MessageBody (string)
* SentOnDate (DateTime)
* SeenBy (List<User>)

**GeoAddress** – represents a real-world location used when offering and searching for journeys.

* GeoAddressId (int)
* Order (int)
* AddressLine (string)
* Latitude (double)
* Longitude (double)

**ChatMessage** – represents an instant message sent between two users using the instant-messenger feature.

* ChatMessageId (int)
* SenderId (int)
* RecipientId (int)
* MessageBody (string)
* SentOnDate (DateTime)
* Read (Boolean)
* RecipientUserName (string)
* SenderUserName (string)

## 2.2 Function Definitions

Below is a list of functions that have been identified as the most important in the sulution. The below list is a subset of all functions present within the system. For a complete list, please refer to the appendices //TODO

|  |  |
| --- | --- |
| Web Service | Android Application |
| 1. UserService   - Registration.  - Auto-Login  - Manual-Login  - Get User  - Update User | 7. GeoCoderTask  - Decode Address  - Decode Latidude And Longitude |
| 1. JourneyService  * Create Journey * Update Journey * Get Journey * Get Passengers * Get Requests |  |
| 1. JourneyTemplateService  * Create Template * Update Template * Delete Template * Get Template |  |
| 1. JourneyRequestService  * Send Request * Process Decision * Get Request |  |
| 1. SessionManager  * Generate New Session * Validate Session * Invalidate Session |  |
| 1. NotificationManager  * Send Instant Message * Send GCM Tickle * Create App Notification * Forward GCM Notification |  |

|  |  |
| --- | --- |
| **Registration** | |
| **Objective** | Users should be able to register in order to access the system using their account. |
| **Parameters** | UserDTO |
| **Action** | 1. User clicks the “Or create a new account” button. 2. Users enters their desired username, email address and password of their choice twice for confirmation. 3. User clicks the “Create account button” 4. Android Application sends request to the WCF service to register a new account. 5. WCF service validates the data. 6. WCF service creates new User object and stores it in the Users table. 7. The WebSecurity module encrypts user’s password and stores it in a separate table. 8. WCF sends a reply to the Android client informing of a successful registration. 9. Android application logs the user in automatically. |
| **Requirements** | 1 |
| **Pre-conditions** | * User must install the application on their Android device. |
| **Post-conditions** | * New user account will be added to the database and the user will be automatically logged in. |
| **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** | JourneyDTO |
| **Action** | 1. User clicks the “Offer” button in the main menu. 2. User specifies departure and destination points for their new journey as well as any optional waypoints. 3. User clicks the “Next” button. 4. Users fills in additional journey information. This includes:    * Departure date.    * Departure time.    * Private    * Allowed Pets.    * Allowed Smokers.    * Vehicle Type.    * Available Seats.    * Fee.    * Additional comments. 5. User presses the “Offer Journey” button. 6. WCF validates the journey model supplied by the user. 7. WCF saves new journey in the database making it available for other users to see. |
| **Requirements** | 2 |
| **Pre-conditions** | * User must be logged in. |
| **Post-conditions** | * A new journey will be added to the database and visible in search results to other users. * 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** | * Users clicks the next button without specifying departure and destination points. * Users clicks the “Offer” button without specifying at least one of the following: * Departure date * Departure time * Vehicle type * Fee |

# Design

This chapter focuses on the design of the entire solution in terms of user interface design as well as system architecture.

## 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 merely a prototype which served as a basis for development of the final designs featuring a much higher level of sophistication and increased user-friendliness.  
 **//TODO insert reference to appendix with all UI sketches.**

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. For all these reasons, 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 had to be performed in terms of screen proportions i.e. 80% of screen width instead of fixed values to ensure proper scalability and positioning of individual components. 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.

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 key to success was design the user interface in way that is intuitive and informative to novice users and efficient for the expert users. The Android application provides features such as input validation and help hints which the novice users will find extremely helpful.

3.1.1.3 UI location and accounting for motion

Most users hold their mobile devices with their thumb acting as the primary finger for touching buttons and other user interface elements. This means that reaching user interface controls on the bottom of the screen and on the right hand side is generally easier and less intrusive for the user. In order to improve user-friendliness, I have placed the user interface elements for the most commonly used functions near the bottom of the screen and on the right hand side to allow for easy access.

The location of buttons was not the only concern and accounting for motion was just as equally important. User’s thumb has a specific flow direction.  
**//TODO insert reference to user interface screenshots in the appendix.**

### 3.1.2 Android Application.

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.

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 had to be designed and positioned with touch-friendliness and usability in mind while conveying the information in the best possible way.

The purpose and meaning of each individual textbox and button control had to be clear right the second it was seen which means that each user interface components visible on the screen had a specific task for which it was designed.

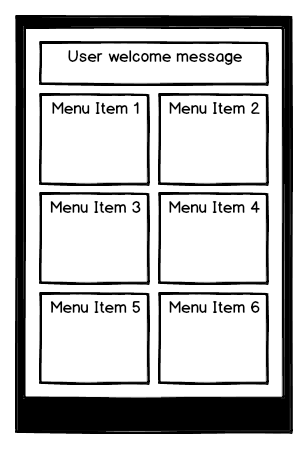


Figure . Initial home activity sketch.

Figure 1 demonstrates the initial sketch for the user’s home activity which they will be greeted with after successful login. The large buttons provide quick and easy access to the main features of the application while making effective use of the available screen space. Just by glancing at the screen for a fraction of a second, the user can decide which section of the application they wish to navigate to thanks to large and touch-friendly button controls. For the list of all initial sketches, please refer to the appendices. //TODO – insert appendix reference.

As far as the application’s colour theme is concerned, it has been decided that a theme which is based on only a small set of colours should be used for consistency 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 - Blue //TODO – insert RGB.
* Buttons - Grey //TODO – insert RGB.

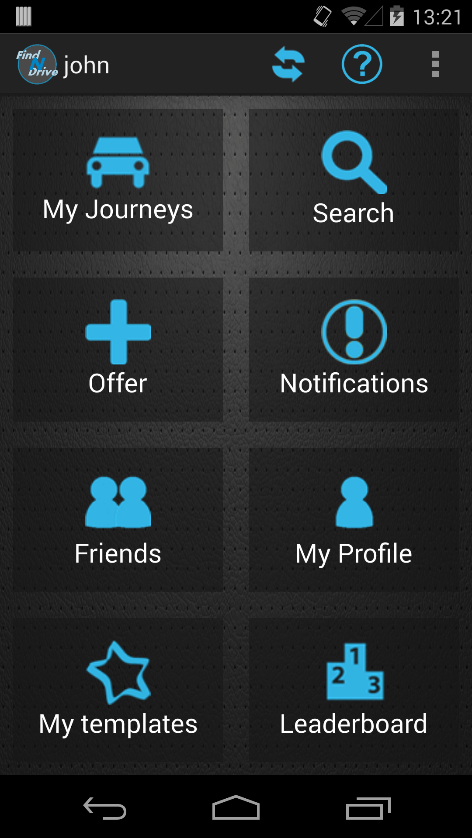


Figure . Final design of the user home activity

* Test - Almost White //TODO – insert RGB.
* Background Image – freely available texture.

#### 3.1.2.2 Final design.

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

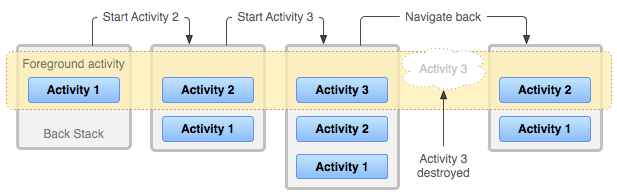
This particular combination of colours reduces the eye-strain due to domination of dark shades of grey which have been proven by various studies to have less negative effects on the eyes.

The final design offers simple yet attractive

It offers 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. For a complete list of screenshots from the Android application, please refer to the appendices. //TODO – Insert appendix reference.

### 3.1.2.1 Navigation

Navigation and screen management in Android is based on a simple yet very effective principle. Activities (screens), 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. //TODO insert figure no. illustrates how the activity stack operates.



Source: https://developer.android.com/guide/components/tasks-and-back-stack.html

The Android application features a very simple navigation system. After user launches the application, they are greeted with the login screen which prompts them to enter their username and password. On successful login, user is transferred to the home screen, //TODO – insert figure no of the above screen, from which they can access all of the functions the application has to offer.

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

//TODO insert navigation map.

3.1.2.2 User Interface Elements

## System Software Design.

* + 1. Top-Level Components.

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

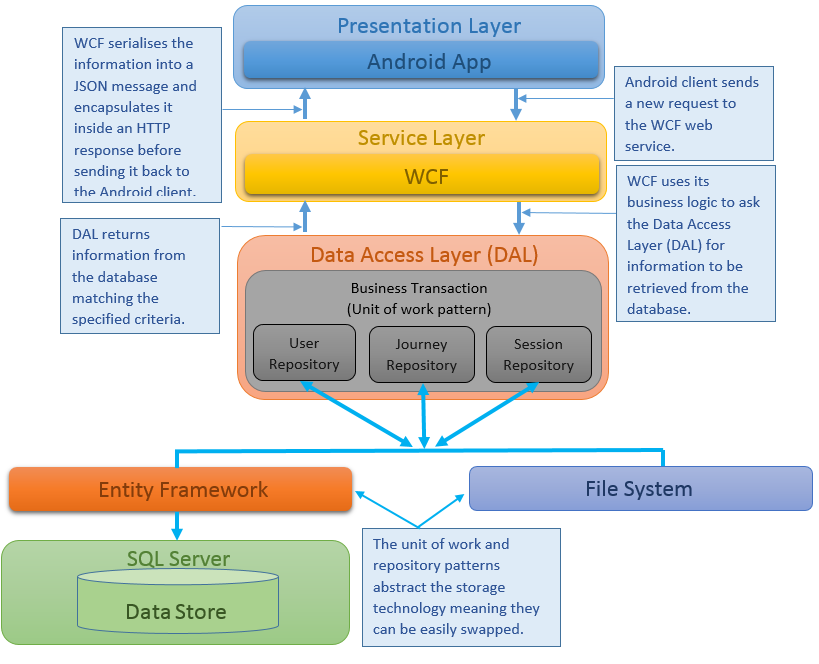
* Database
* WCF Web Service
* Android Application
* Google Cloud Messaging Servers (GCM)

//TODO insert network diagram.

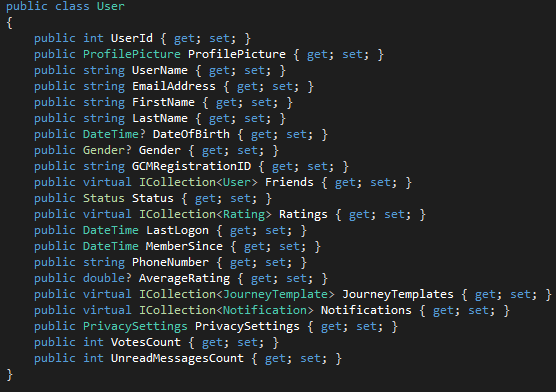
* + 1. Multi-tier architecture.

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.

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.

//TODO – insert figure no.

* + 1. Database Design

The application’s database has been designed using Entity Framework’s code first approach. The code first approach uses application’s model objects to generate the entire database without the need to ever open the database designer. The code-first approach offers a number of advantages over the traditional database design. One of them is the more code-centric approach in defining of the database schema. This is due to Entity Framework’s ability to generate the database model from ‘plain old classes’ with no base classes required. As an example, consider the following class representing the ‘User’ object.

//TODO insert figure no.

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

//TODO insert the schema.

* + 1. 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. To achieve this, 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 UserService 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 //TODO – insert figure no. which illustrates the interface from which UserService is implemented. It clearly shows that all the methods located in UserService concentrate on the management of user accounts. For a complete list of UML diagrams of all service interfaces, please refer to the appendices. //TODO – insert appendix reference.

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, ie. 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.

* + 1. Security

Being of a paramount concern, security is a vital part of the project’s planning and development processes.

* + - 1. Transport Security

Transport security has been achieved with HTTPS,

* + 1. Google Cloud Messaging (GCM)
    2. Google Maps