Lost & Found

Conor Smyth - 12452382

Adam O'Flynn - 12378651

Supervisor: Brian Stone

Technical Manual

Contents

1. Introduction

Our project is a geo-location lost and found iOS application. Users are able to post on the application that they have either lost or found an item. This is called a listing. A listing is made up of information like a title, lost or found status, a category, a description, the location of where the item was lost or found, and an image of the item. One of the key features of the application is the ability to drop a pin on a map where the item was lost or found. Once a user places a listing, if they search the map, they will be able to see their listing as a pin annotation on the map, along with all of the other listings. Users can search the map or use the category search to find listings.

The application acts as a medium of contact between people who wish to find an item they have lost or for people who wish to return an item that they have found. The listing page supplies an email address and an optional number for other users to enquire about that listing.

The application has a number of different functions.

- To use the application, the user is required to log in.
- The application will react to your location to see what listings are near you. If the user does not wish to reveal their location, they can deny location services for the app.
- Users can create a listing.
- When creating a listing users must provide some information about the item, have the option of a picture and the option of dropping a pin on a map to display exactly where the item was found.
- Title, description, category, and status are required information for a listing.
- Users can look for listings around them on a map. Listings are displayed as pin annotations.
- Users can upload a photo to use as their profile picture.
- Users can report a listing.
- Users can remove their listing once resolved.
- There is a categories page where listing will be split up into their respective categories allowing for users to easily search for specific listings.
- Users can contact the owner of a listing.

The application uses the Parse Framework as a cloud backend database. Parse supplies a free database in which the application can save and retrieve data with a REST API. Parse also handles user management like password hashing, checking for duplicate usernames, emails and also email verification.

1.1 Glossary

Parse - The Parse Framework is the mobile backend service that the application uses.

Mobile Backend as a Service - A model to allow developers to connect their apps to backend cloud databases, APIs, and, in our case and also provide user management.

iOS - The apple mobile operating system on which the app runs.

XCode - The Integrated Development Environment (IDE) in which the app was built. XCode provides the tools required to run and build iOS applications.

Git - Git is an easy to use version control system we used in development of the app.

Storyboard - The Interface Builder in XCode which the app was made in. The Storyboard is used to make User Interface and different views for the app. The storyboard simplifies the design of apps.

View and View Controller - Views show and handle the application's user interface. View Controllers link the application view and the data.

Segue - This is the name for the transition between two view controllers on the storyboard.

Objective C - The programming language in which iOS apps are developed.

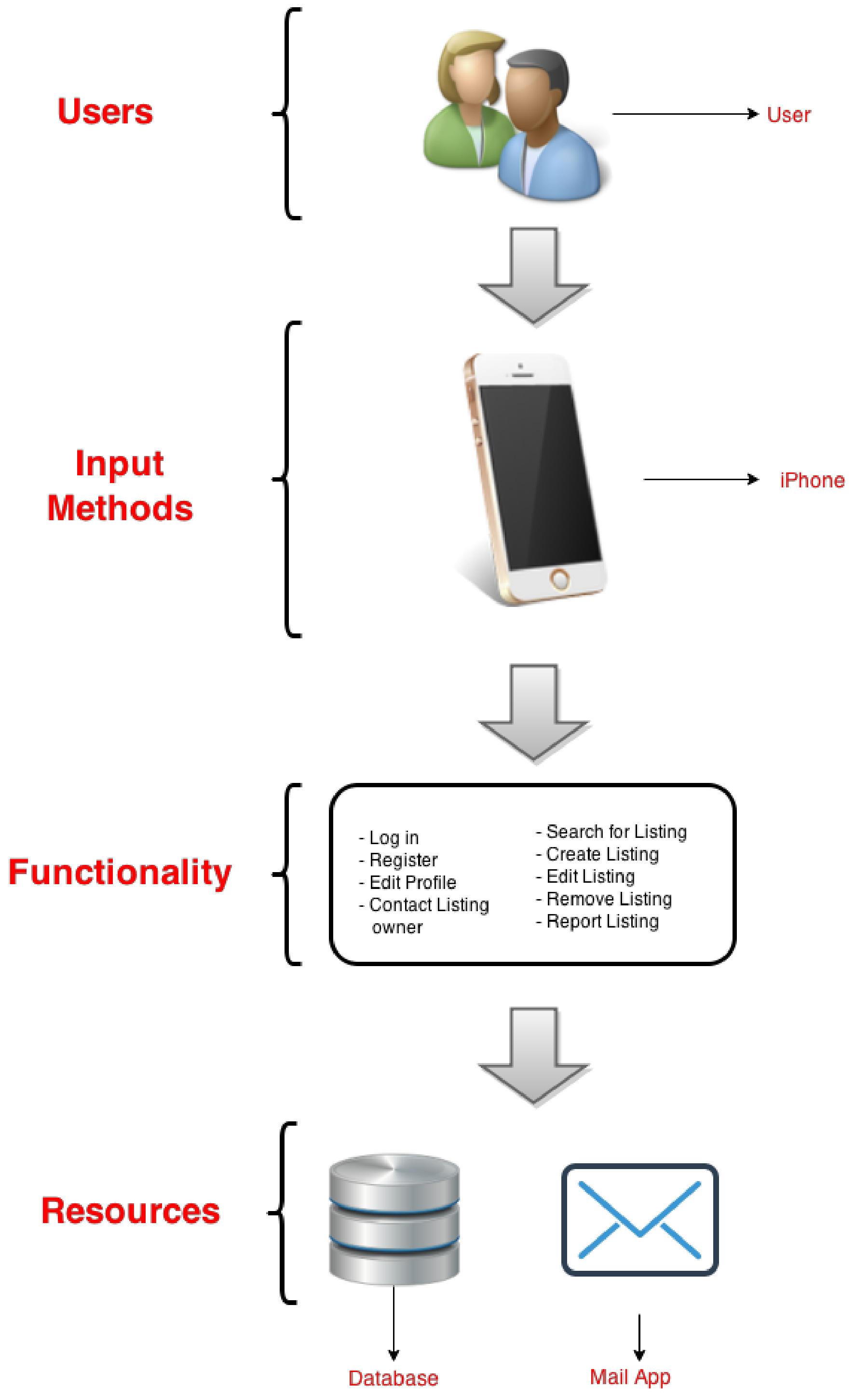
Use Case - A use case is a diagram or description of how a user interacts with the system for certain scenarios and shows what functions the user will use in order to reach a goal for that use case. For a use case diagram the actor is shown as a stick man. The actions are shown using ovals with connections between showing sequence. The use attribute is used to describe a function that occurs as a result of another action or function and extends attribute is used to describe the extension of another function or action if a certain condition occurs.

REST API - REST is an acronym for Representational state transfer. REST supplies a uniform interface between an application and another system, in this case our application and the Parse database. Using a RESTful interface adds scalability to the application so that other features can be easily added such as a website or other systems. REST makes use of how HTTP handles resources using keywords such as:

- GET: request data.
- PUT: update data.
- POST: create new data.
- DELETE: remove data.

2. System Architecture

2.1 Architecture Overview Diagram



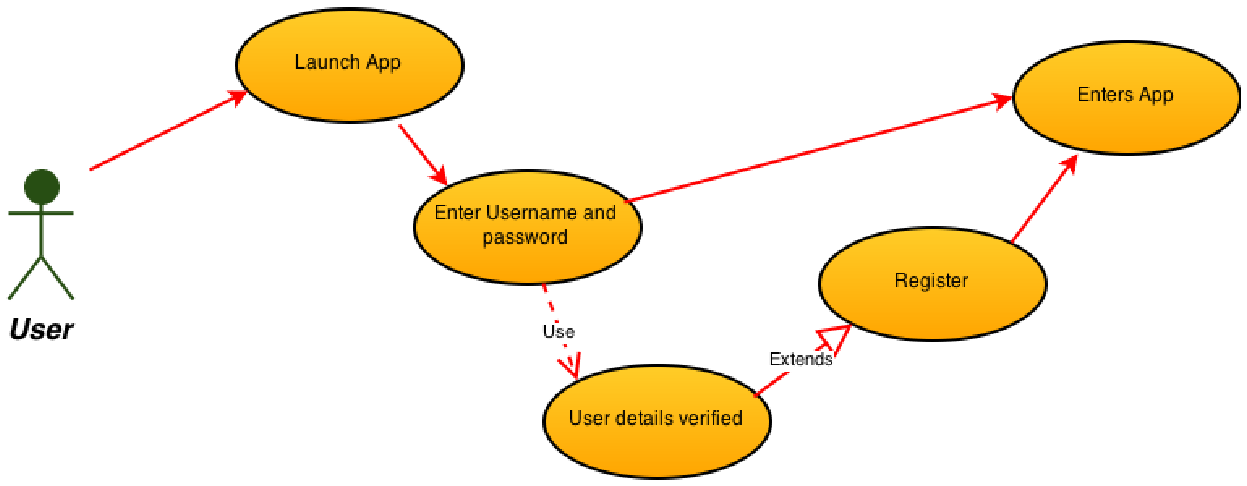
Here is the architecture overview diagram for the system. This diagram gives an easy to understand overview of what users will be using the system, how the users will use the system and what resources the users access while completing the functions of the system.

The diagram shows the Users of the system who are our targeted users. These users will use an iPhone as their input method. The users can then use the functions shown to access the resources.

2.2 Use Case Diagrams

Here are three architecturally significant use cases to demonstrate how the system flows from the user to the resources on the architecture overview diagram for key requirements of the system. These use cases are architecturally significant because they show how the system meets the key functionality of the application.

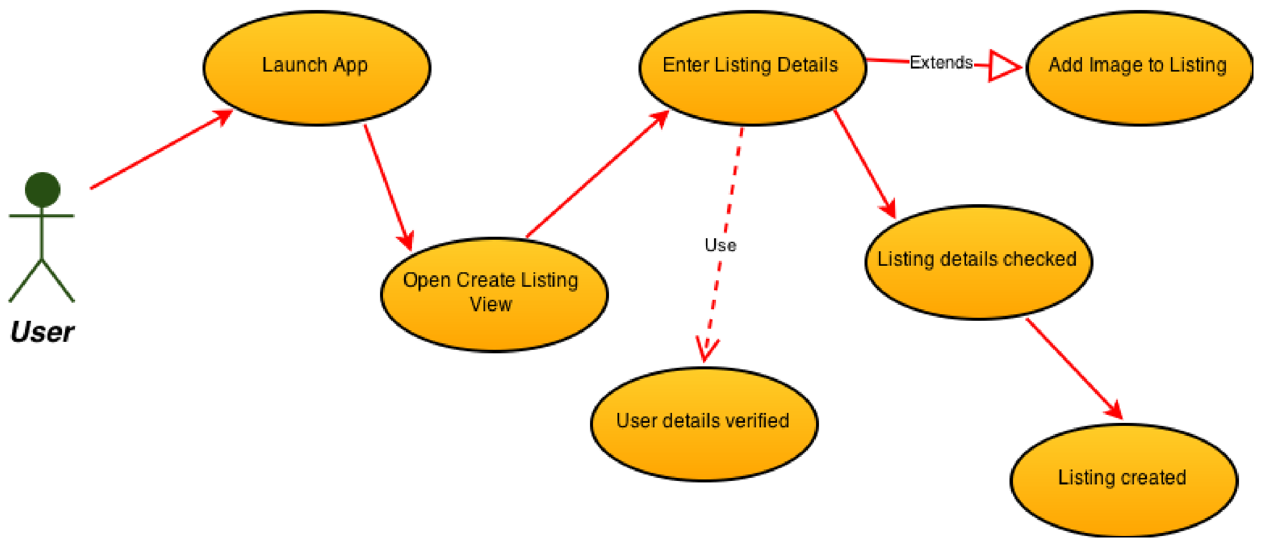
2.2.1 Login



This use case is for a user to log in. The primary actor here is the user. Here the user launches the application and enters their details. The details are then verified. If the user's details are not there they will be requested to register otherwise they may enter the application.

This use case shows how the user interacts with their iPhone to use the login function and access the database resource to verify the details or register their details.

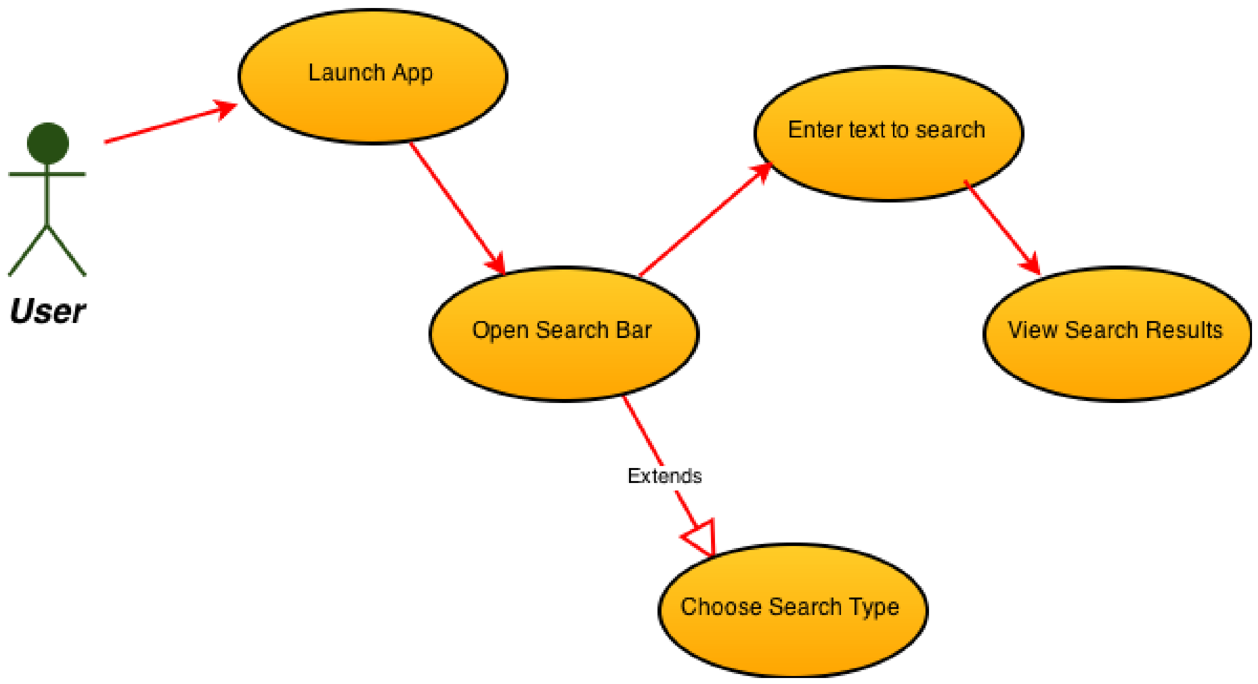
2.2.2 Create Listing



The next use case demonstrates creating a listing. The primary actor here is the user. The user launches the application and selects the option to create a listing. The user then enters the listing details. These details include a title, description, date lost/found, category and potentially an image or location. The user is then checked, to see if their email address has been confirmed. If the users email is not confirmed they cannot create a listing. The details of the listing are then checked to ensure the user has chosen a category and not left some fields blank. The listing is then created in the database.

The above use case demonstrates how the user interacts with their iPhone to use the create listing function and then uses the database resource to create a listing on the database that can be accessed by other functions such as the search for listing function.

2.2.3 Search for Listing

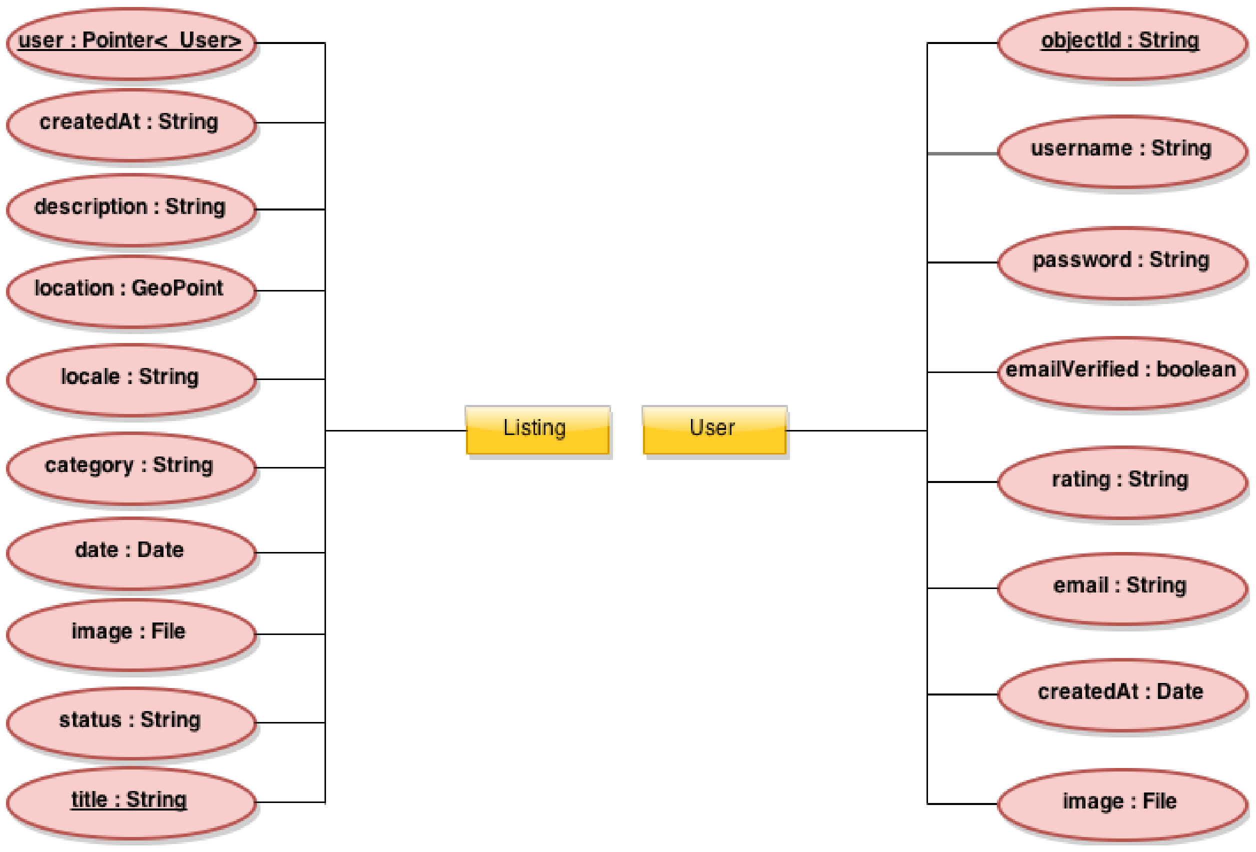


Here is the use case for a user searching for a listing. The primary actor here is the user. The user launches the application and opens the search bar. The user then enters the text that they wish to search for. The user can chose a search type if they wish but it is not necessary. The search types are all, lost or found which filter the results to what the user desires. The user is then redirected to a view that shows them the results of the search from the database.

This use case demonstrates how a user using an iPhone uses the search for listing function to access the database resource and view the results for the search.

2.3 Entity Relationship Diagram for Database

The structure of the database was chosen through developing the architecture overview diagram and use cases to determine what objects were necessary to be stored for the system to function with the key functionality.

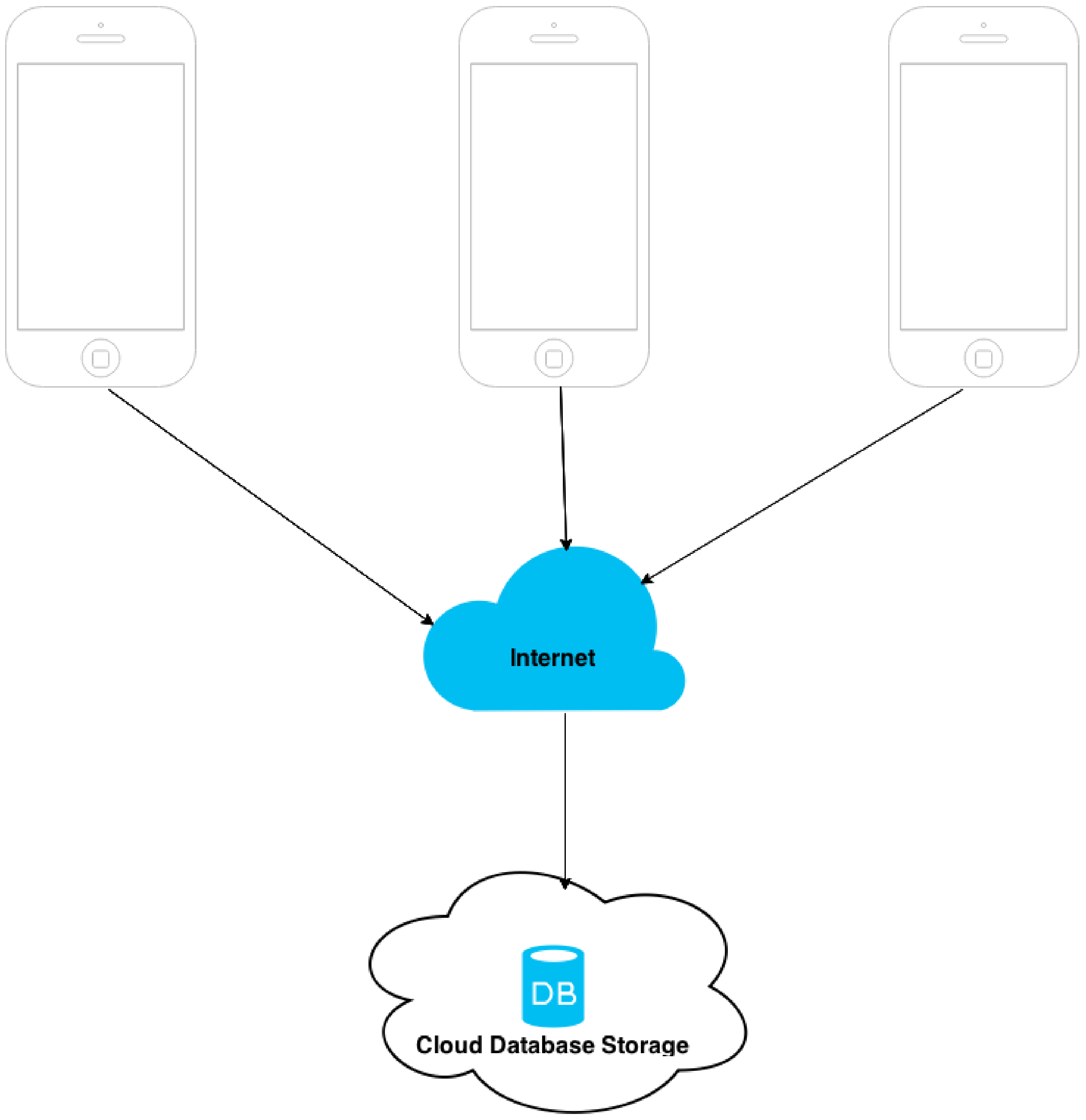


Here we have the Entity relationship diagram for the database. This diagram shows the data and how it is related to the entities of the database. The database consists of two classes, Listing and User. These classes hold all the information relevant for the system to function.

The Listing class has numerous objects that hold the information that a Listing requires for the system. The String title is an identifier object that we will be used to return search results when a user searches for an item. The Pointer<_User> user object is another identifier object that is used to associate a listing with the user that posted the listing so that the Listing view can be used to have a link to the posting user.

The User class has numerous objects that hold the information that a User has for the system. This information is retrieved from the user at registration. The String objectId is an identifier object used to differentiate between users.

2.4 System Architecture Diagram



Here is the high level system architecture diagram that how our system architecture is composed. The diagram shows multiple users from their devices connecting through the internet to the cloud database storage.

The users will be using iPhones running iOS 7 or iOS8. These are the two main operating systems that iPhones use and the operating system that the application is developed. This operating system was chosen because of the increasing number of iOS devices and this is the operating system that these devices use.

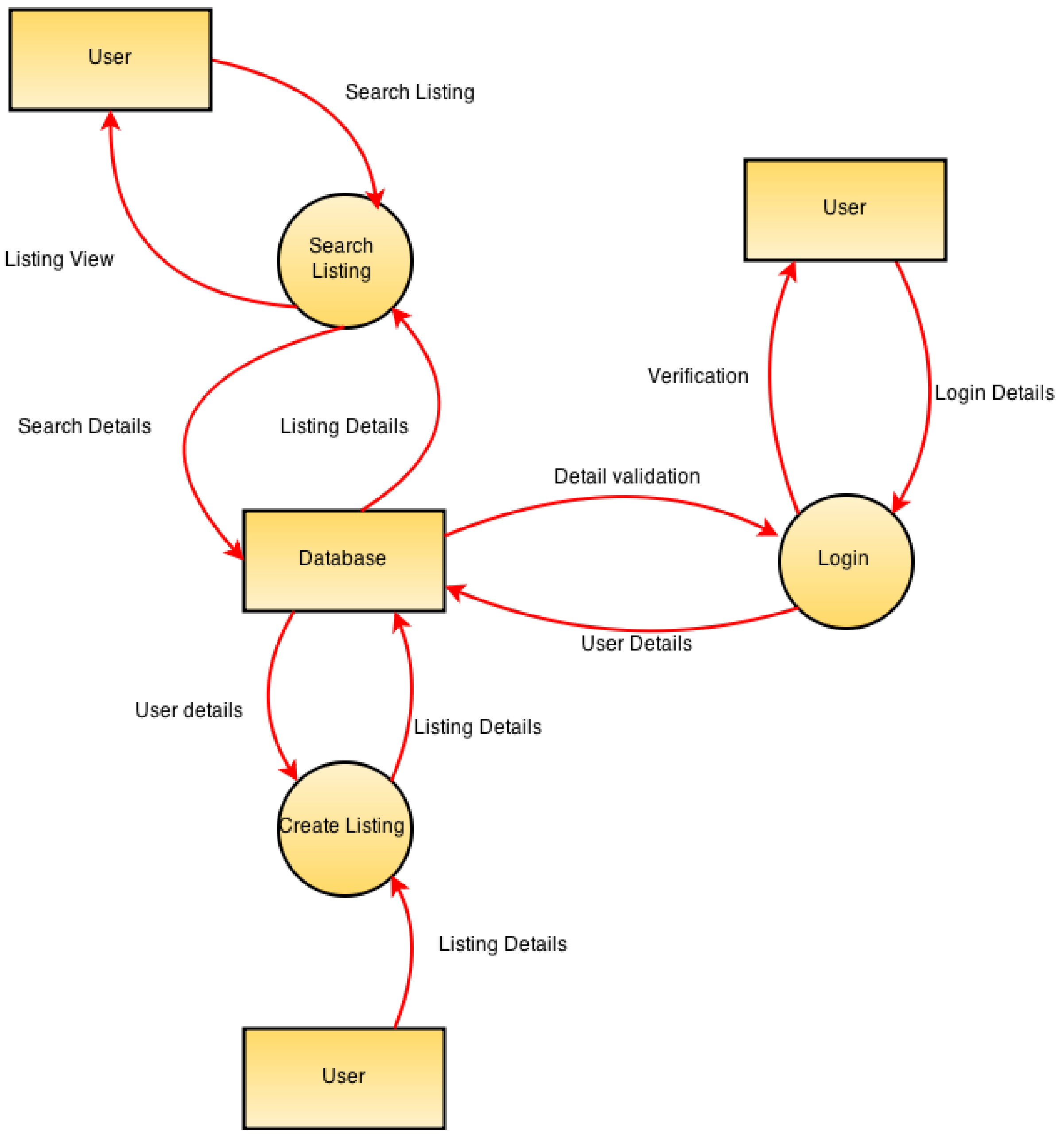
The users will download the application and connect using the internet to the cloud database storage. Through the Internet the user can request for data through searching, creating and viewing listings.

The database is a cloud storage framework called Parse that handles all the queries using REST API allowing for easy to construct requests in order to retrieve data This gives the option of scalability for a website or another application to retrieve data from the database..

3. High Level Design

3.1 Data Flow Diagram

Here is the Data Flow Diagram developed through of the system architecture and use cases.



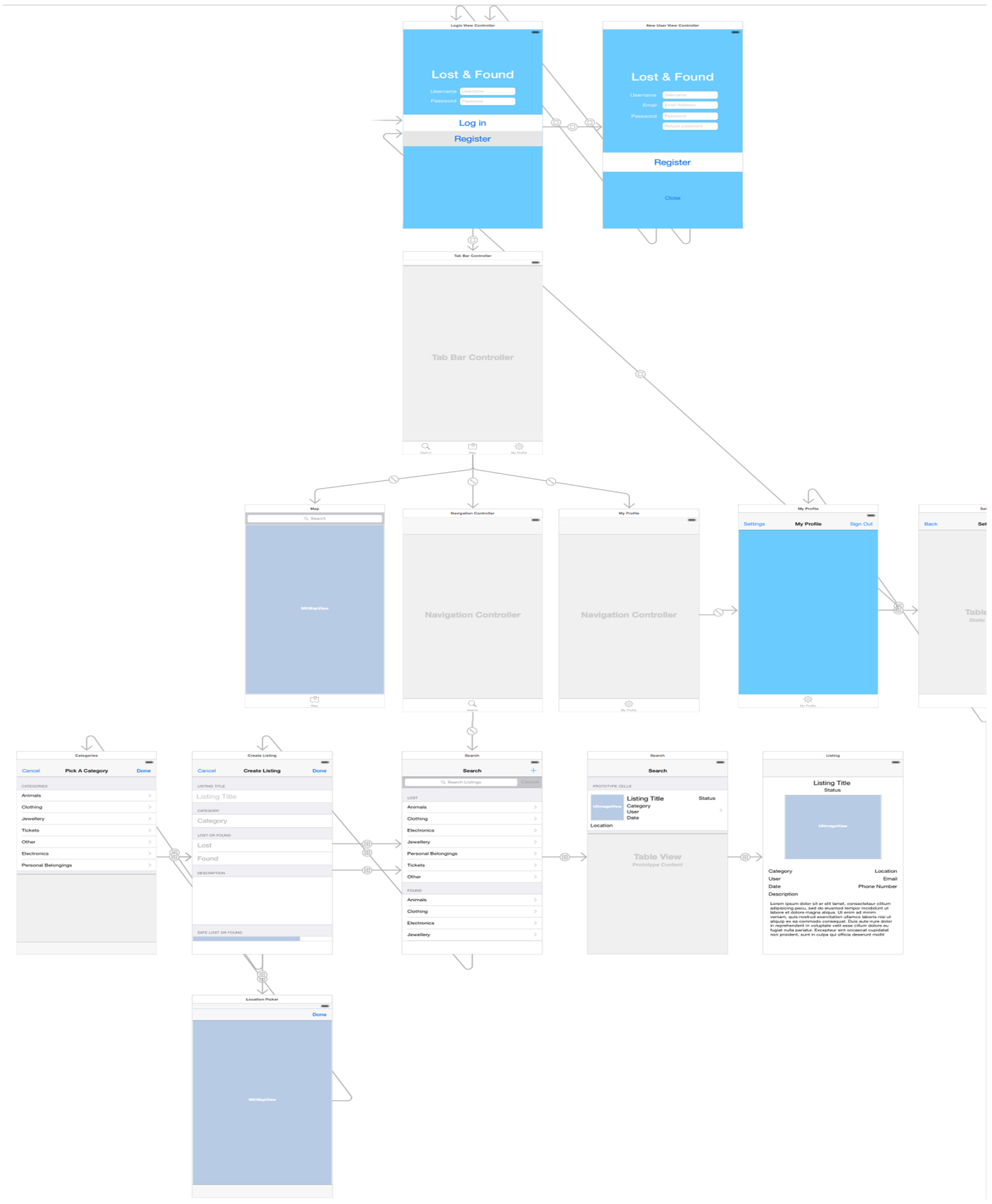
This diagram shows the flow of the data for the Log in, create listing and search listing use cases.

When the user logs in the data is sent to the database which then replies with validation data with the information on whether the user is valid.

When the user creates a listing the data the user enters gets sent to the database and the database replies with information on whether the user has a confirmed account.

When the user searches for a listing the text they have entered gets sent to the database. The database then replies with the information that matches the search and the information is shown to the user.

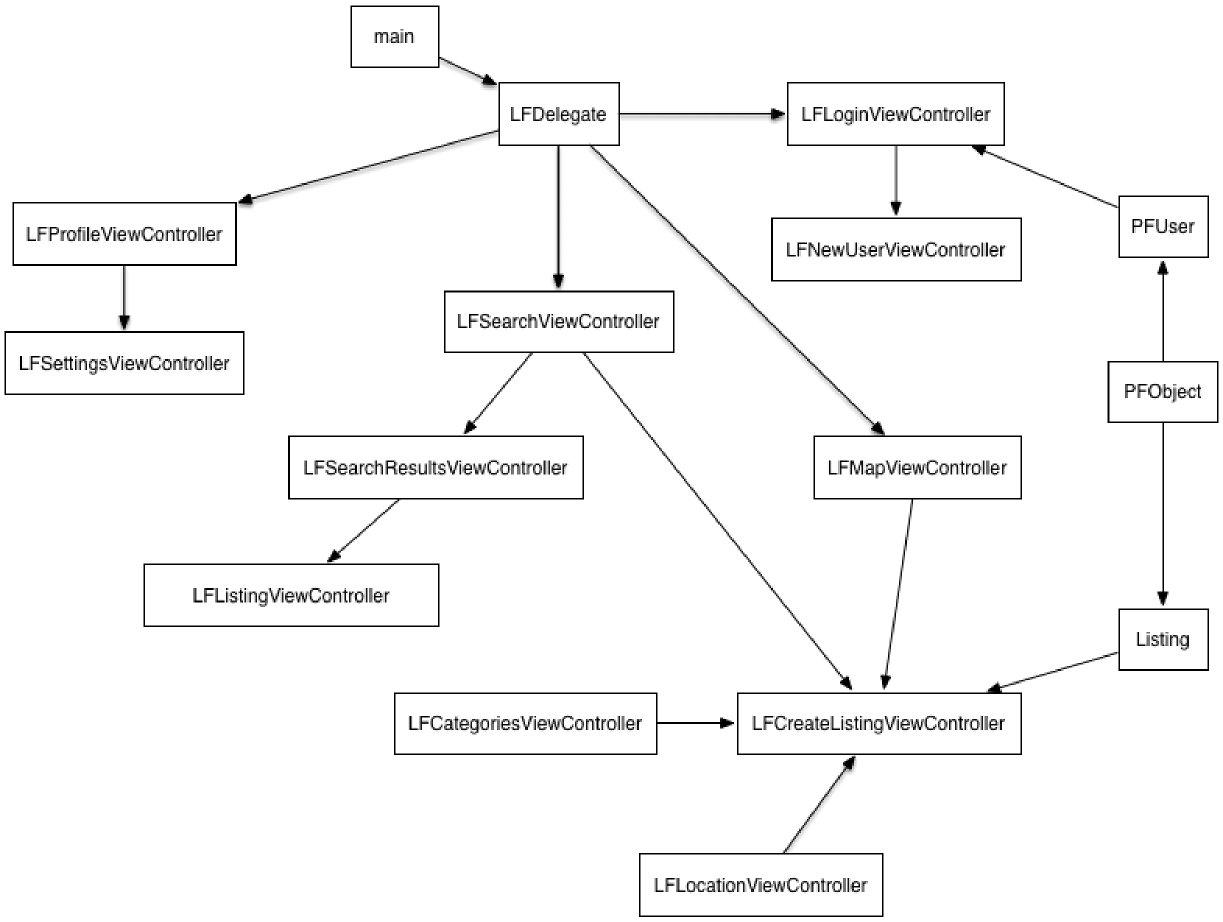
3.2 Storyboard Interface of Application



This diagram of the storyboard shows how the individual views interact and connect with each other. The links going from one to the other are the segues. The diagram shows the hierarchy that the views follow. This was very important in developing the app. Once we had prototyped the app through the storyboard, we could add the code behind the controllers. The storyboard helped us add UI components like buttons and let us program the methods behind them.

3.3 Dependency Model

Here is a dependency model for the system. Through the data flow diagram and development of the system architecture this dependency model was developed to show classes of the system and how they depend and interact with each other. This model shows the classes that correspond to the Storyboard view shown above.



The main class is the class the runs the thread that the application runs off. This thread runs the LFDelegate which is backbone of all the other classes and handles the returning user function so a user doesn't need to constantly sign in if they have previously.

The PFObject class is the backbone class for all Parse objects that the system is based off. The Listing and User classes are extensions of the PFObject class.

If the user wishes to login the LFLoginViewController is used to show the login screen and the class uses the user class to determine the user details. If a user isn't registered they are redirected to the LFNewUserViewController class that handles new users.

The LFProfileViewController and LFSettingsViewController classes handle the user profile and the user settings pages.

The LFSearchViewController class is related to many other classes depending on what function is required. For searches the LFSearchResultsViewController class displays the results which can be inspected using the LFListingViewController class.

If the user creates a listing the LFCreateListingViewController class is used to retrieve the details and this class uses the LFCategoriesViewController class to choose a category and LFLocationViewController class to choose a location if needed.

Through the use of these classes the implementation of the system meets the functionality specified by the requirements of the system.

4. Problems and Resolution

4.1 Using the Interface Builder

To create the user interface and the view controllers, we used the storyboard. This piece of software allowed us to quickly prototype the application. Adding the code to the view controllers and connecting them through segues in the storyboard proved very difficult at the beginning. Neither of us had any experience in application development and iOS user interface development. Through tutorials and YouTube videos, we quickly learned how to use the storyboard. This was a very helpful tool in the design process as it allowed both of us to visual our ideas. Once we overcame the initial learning curve, we strived in using the interface builder and regarded it as a very powerful tool. Since the storyboard handles the views, you did not need to hard code in basic UI components like tables, buttons etc. We just added these and programmed the methods behind the UI.

4.2 Programming in Objective-C

Programming in a completely new language is always challenging. Syntax-wise, objective-c was very different to any other language we have learned. Connecting the views from the storyboard into the view controllers with code was very hard at the start. Programming custom segues to move data between views proved to be very intensive and took us around two days to fully understand and implement them. We used tutorials to learn the syntax. After the initial delay learning objective c, we didn't have many other problems.

4.3 Developing the App on Mac

One big problem with developing our application was that we had to use OSX. With neither of us owning a Mac, we had to use virtual machines with OS X installed on them. Since we were using virtual machines, the operating system could be quite slow. This really hindered our development. To improve this, we allocated most of our RAM to our virtual machines and didn't use our host operating systems at all during development.

4.4 Releasing App on the App Store

To release an application on the App Store, Apple requires you to have an Apple Developer Licence. This licence doesn't come cheap at €99 a year. Not only does this not allow us to put the app on the store but you cannot test apps on a physical device without this licence. The only way we could test our application during development was to use the built in iOS simulator in XCode. This simulator was great, but doesn't provide us with actual feedback of performance on a device.

4.5 Transferring Data between Controllers

When we were working on the create listing aspect of the application, we found it difficult to go to a view and send data back to the source controller. We looked online for a way to do this, but most of these ways needed us to hard code in showing the views. This required us to not use the storyboard. To fix this, we discovered a way of “unwinding” a view. This segue would exit the view, and with a bit of coding, would allow us to pass data back without refreshing the original view controller. This is well-commented in the source code.

4.6 CoreLocation framework with iOS8

When developing the map interface for the system we ran into difficulties with new features implemented by apple in their newest OS, iOS8, which we are developing on. It was not very clearly documented by apple and resulted in multiple problems when requesting permission from the user to access their current location. The problem occurred where an extra field needed to be added to the info.plist file which dictates some key features that an application being developed requires to function as expected.
The added field was NSLocationWhenInUsageDescription with the text supplied being what prompts the user. In previous versions of iOS, simply calling the function to start updating the user's location was sufficient for the OS to prompt the user for permission to use their location. However after some research we discovered that this problem was widespread among iOS8 developers catching a large number of people out who knew no way to fix this issue. Eventually we discovered this extra key, NSLocationWhenInUsageDescription, in the info.plist was required to ensure the prompt shows. This allowed us to prompt the user for permission to access their location when the application was in use. We chose this option as we have no need for location services if the application is minimised.

5. Installation Guide

5.1 System Requirements

5.1.1 System Requirements to Run the Application on Mac

- iOS applications can only be built and ran on Apple Machines.
- To run the application smoothly, you will need to have OSX Yosemite.
- To run the application on a mac, you will need to download the latest version of XCode. This is available from the app store and you will need a valid apple ID.
- If you already have XCode, you will need to update it to XCode 6.2 with the iOS 8 SDK.
- To update XCode, open the App Store and check for updates.
- All required frameworks are supplied in the project folder and will automatically be loaded in when the project is opened.

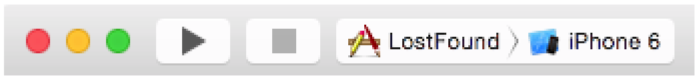
5.1.2 System Requirements to Run Application on a Device

- You will need an iOS enabled device running iOS 7 or later to run this application.
- You will require an internet connection to download the application.
- You must have a valid apple ID to download the application off the app store.

5.2 Installation Guide

5.2.1 How to Install the App on Mac

- To run the application, you will need to start XCode. If you do not have this program, the system requirements, above, describe how to get it.
- You can open the project in a number of different ways.
- In Finder, you can open the project from the .xcodeproj file in the project folder.
- In XCode, you can drag the .xcodeproj file into the IDE.
- In XCode, you can also go to the File section at the top of the program and open the .xcodeproj file from there.
- Once you have the project loaded into XCode, you can build and run the application by pressing “cmd + R”. Alternatively, you can press the “play” button at the top left of the screen. To make sure the right system is running, make sure the iPhone selected is on the iPhone 6, like below.



- When the application is finished building, the iOS simulator will appear.
- You can now use the application like you could on a physical device.

5.2.2 How to Install App on iPhone

- To download the application, you must go to the app store.
- In the store, search for the application using “Lost & Found”.
- Once you find the correct app, click “Get”.
- You will be prompted to enter your AppleID password.
- If it’s valid, the download will begin.
- Once finished, go to the home screen, where you will find the application.
- Run the application.

6. References

Parse - https://parse.com

iOS Developer Library - https://developer.apple.com/devcenter/ios/index.action

Project GitHub Repository - https://github.com/Adam-Conor/project_ca3

Use Case Information - http://en.wikipedia.org/wiki/Use_case