TECHINAL SPECIFICATION

CA400

Student Name: Cian McCann

ID: 11482422

Supervisor: Ray Walshe

ABSTRACT

This project aims to provide a cost free alternative to the Community Text Alert Scheme. It is made up of a cross platform mobile application and a desktop web application. The mobile app allows users to create and verify a user account to receive notifications of crime or suspicious behaviour. The desktop application can create reports and notify users based on their GPS location. Users of the mobile app can control the reports they receive using their account preferences.

All of the source code for the Community Area Alert mobile and desktop application can be found in my repositories on GitHub:

https://github.com/Cian500?tab=repositories

TABLE OF CONTENTS PAGE

MOTIVATION

I originally became interested in the area of developing an alert system for An Garda Siochana when I noticed a post from the An Garda Siochana Facebook account on my timeline. The report was a description of a young boy who was abducted in Cork, I thought it was strange how an organisation like An Garda Siochana was using Facebook to send out alerts which relied on members of the public to share their posts so word could be spread. It was almost 5 hours after the post was created when it finally appeared on my timeline, by which time it seemed too late to share the post.

Once I felt I had discovered the need for a Garda alert system I began to examine different possibilities about how the system could work and be implemented. I originally proposed a concept which would involve users creating reports which could be sent to other users, I felt this had the ability to become a good mobile application but in terms of a fourth year project I wanted something more. I then noticed a sign about 100 meters from my house declaring that this is a "Community Alert Area", I had passed this sign almost every day for the past 7 years since I moved house but I never really took any notice of it until that moment. The sign clearly stated that my house was a member of this "Community Alert Area" however despite living there for 7 years my parents and I had never heard anything about this scheme or received any alerts. This ignited my interest in the Community Area Alert scheme and lead to me researching the idea more.

After researching I felt I had come up with a concept which could make a good final year project and might also be capable of replacing the current method of sending out community alerts. This idea is a lot more complex than the original concept of users sending reports to other users, this made me more motivated to develop this idea compared to the original concept. The original idea would have produced a decent app which may have received a few hundred downloads, but the new concept was a lot more motivating in that it is all or nothing. The Community Area Alert System I developed will either be used in Garda Stations all over the country allowing members of the public to receive notifications whether they are in a Community Alert Area or not, or this project will never be used unless I redesign certain aspects. This gave me increased motivation to develop the Community Area Alert system to a high quality which meant could put into practice everything I had learned over the last 4 years in DCU.

In the next section I will document the research I did before deciding on the functional requirements of my new system. I will record some of the interviews and discussions I had as well as links I discovered which outlined some of the issues with the current Community Area Alert scheme and how it could be improved. All of this research increased the chances that my system might actually be implemented which increased my motivation. I also discovered the positive effect the Community Area Alert scheme is having around Ireland. It motivated me to think that if my system could replace the current alert scheme that it would be able to reduce crime and help people to feel safer at home.

This all increased my motivation for developing the Community Alert system but of course my main motivation was to complete my fourth year project by doing something which allowed me to use what I learned throughout my four years in college and gain new skills along the way. This idea provided me with a good platform to achieve these goals. The main languages involved are HTML, javascript and CSS, these 3 languages were identified as a weakness of graduates from Computer Applications by one of my lecturers which gave me increased motivation to learn as I was developing the Community Area Alert system.

RESEARCH

Once I had become interested in the Community Area Alert scheme I began research into the current system and how it operates. Straight away I began to realise just how popular text alert schemes have become in rural areas in Ireland but I could not help but notice how complex the current scheme is. I discovered the basic objective of the scheme is to:

"Reduce the amount of crime and reduce the fear of crime."

I liked the idea behind it and became interested in learning more. The idea behind setting up a text alert scheme is positive and seemed to be working well, unfortunately from what I had read the process seem rather inefficient. The following link explains how once a Community Area Alert scheme is set up by a community, one member of that community holds a specific phone and is responsible from alerting neighbours when asked to do so by the Gardai. The link explains how this process can take 20-30 minutes to alert everyone in the area. It also explains how problems arise if the person in charge of the phone does not keep it on their person at all time.

http://www.boards.ie/vbulletin/showthread.php?p=89511339

Most of the information I found online was from www.garda.ie so it was hard to get real views on the benefits and drawbacks of the scheme. I therefore decided to take my research a step further. I began by contacting Muintir Na Tire, this is the organisation who are responsible for helping communities set up a community alert scheme. They are the organisation who originally came up with the concept and proposed it to An Garda Siochana. I spoke a lady on the phone who gave me the personal number of Muintir Na Tire's Director of Service James O'Neill. He was very helpful and showed a real interest in the project I proposed. We spoke on the phone for almost an hour, he talked through the scheme in great detail and included everything from setting up to running a scheme. The entire interview is documented in my blog but here are some of the main parts I took from it.

Current State if the Community Area Alert Scheme

James's said setting up a text alert scheme can reduce crime by on average by 24% and increase drug detection by 14%. He also gave me an example from a months ago of how the text alert scheme saved a young boy from committing suicide in a forest.

Setting up a Text Alert Scheme

I was surprised at the cost and how much is involved in setting up a text alert scheme. It requires volunteers to call around to each house in the area with a consent form. Any households which choose to sign up to the scheme must be called to a week later by the same volunteer so the form can be collected. A signup fee of 15 euro is also collected, this is to pay for text messages and the Community Alert Area sign which has to be placed in the area. It also costs 82 euro to insure a volunteer. Straight away I could see opportunities to send free notifications and offer a consent form through a mobile app.

Cost of Running

Despite all the benefits of setting up a scheme James said due to costs "the success of the scheme may be its own downfall" the reason behind this is the cost. He said on average a text alert is sent out every 2-3 weeks. If there are 500 households in the area that means the overall cost of sending texts is usually around 360 euro a year, this is on top of insurance and even the cost to

put up a sign. He then said in areas where the scheme is working well it can cost up to 1200 euro a year, he said this is not sustainable. He was very interested in the idea of sending out free notifications which would significantly reduce costs.

Talking to James helped to define the set of functional requirements that the Community Area Alert system needs to accomplish. It also helped me to identify other areas in which my system could improve upon the current scheme.

After I finished speaking to James O'Neill I contacted my local Garda station in Kells county Meath to get more information on what each functional requirement of my Community Area Alert system should accomplish. Most of the information I got form the Guard was already mentioned by James but at least it reinforced what I already knew. After speaking with the Guard I felt like I was in a position to start the functional specification.

I also did research to make sure this type of system was not already available, I realised there is not anything like it at the moment which increased my motivation to work on developing this Community Area Alert system as my fourth year project.

DESIGN DOCUMENT

INTRODUCTION

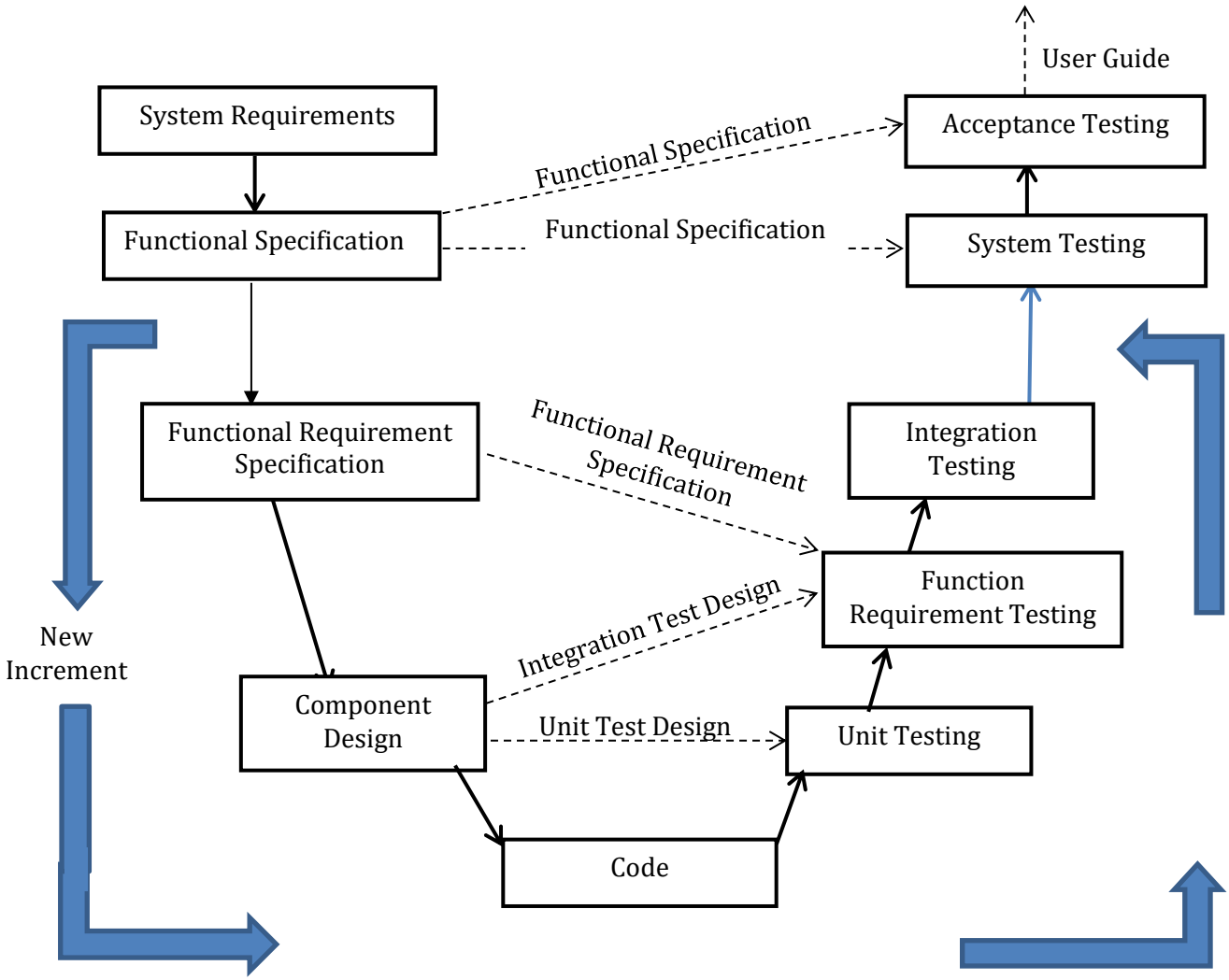
The aim of this report is to describe how I intend to develop the Community Area Alert system. It will provide a building block for developing individual parts of the system as well as the reasoning behind my thinking. It can be used by someone who wishes to modify the system once it has been created or by someone who is studying the system to establish the reasoning behind my design.

SOFTWARE DEVELOPMENT PROCESS

To assist my development of the Community Area Alert system I created my own Software Development Process. I decided to adopt an incremental approach where I will construct a partial implementation of the total system and then add increased functionality. Each cycle of my development process will add a new functional requirement to the system. An example of a functional requirement is User Account Creation, all functional requirements are defined in section 3 of my function specification. In order to achieve this I decided to develop the Community Area Alert System using a modified V-Shaped Model. I felt that combining an incremental model of software engineering with the V model would suit my project for the following reasons:

- A large part of any project involves testing. The V model emphasizes verification and validation of the product by testing in parallel with a corresponding stage of development. Bugs will therefore be found early so they do not flow downwards and cause more issues.
- Since the requirements have been defined, documented and are well understood they are less prone to changes which suits the V model.
- It is essential this system is completed before the deadline so it is important the requirements are defined early and are understood.
- Documentation is an important part of this project. Documentation will be produced at defined stages within each increment of the V model.
- The V model is simple and easy to use especially on small projects. By modifying the V model to make it an incremental development process I should save time overall and still meet all of the systems requirements.

INCREMENTAL V-SHAPED MODEL



System Requirements

This is the first stage in the development cycle where the requirements are understood. This phase involves detailed analysis of exactly what the system needs to achieve from the point of view of the current Community Area Alert system, project supervisor and myself. The information gathered at this stage is used to create the functional specification.

Functional Specification

This functional specification will contain a detailed overview of the Community Area Alert system. The aim of this report is to describe exactly what the Community Area Alert system is required to do and provide enough information on the software tools and packages which will be used so that the system can be designed to meet its requirements. I will provide a detailed description of each component in the system and their requirements which will be used as the starting points of new increments in my system.

New Increment

The highest priority component of the system which has not been developed will be selected and added to the overall system in this increment. Each increment will have its own name and version number.

(Verification Phase)

Functional Requirement Specification

This stage specifies the functional requirement to be added and what the functional requirement should accomplish. Each functional requirement has already been specified in the functional specification. It originally intended to document this stage but due to it not being required I decided to drop the report however I still identified functional requirements at this stage.

Component Design

This stage in my development will contain an overview of both the high and low level design of the particular functional requirement to be developed in this increment. It will include the actual logic such as design patterns, UML diagrams, methods and other components of the system which this new requirement need to be integrated with. Unit tests will be developed at this stage. This stage was documented under the design section of the technical specification and can be seen below. The design of each functional requirement is documented under the name and version number of the increment where it was designed.

Code

This is where all of the coding takes place, a new functional requirement will be added to the Community Area Alert system at this stage. Once coding is complete the development process will enter the validation phase.

(Validation Phase)

Unit Testing

Unit tests designed in the Component Design phase are executed on the code during this validation phase. Only unit tests created for the particular functional requirement created during this increment will be tested.

Function Requirement Testing

This involves black box testing on the component which was created during the particular increment. The component will be tested to ensure it complies with its specific requirements defined in the Functional Requirement Specification document. I will seek to detect defects found within the functional requirement developed during this increment only. Testing will follow the Functional Requirement Test Plan which is explained in the validation plan.

Integration Testing

Integration testing will involve grouping together the component being developed in this increment with all of the other components which it was identified to interact with in the Component Design phase, then applying my integration test plan to the group of components.

End of Increment

At this stage either a new increment of the development process will begin or if all functionality has been added the development process will proceed to System Testing.

System Testing

This will check the entire functionality of the system to make sure it is working correctly on each operating system and that it is communicating with the external database correctly. The functionality of the system will be checked against the requirements in functional specification. See the system test plan for more information.

Acceptance Testing

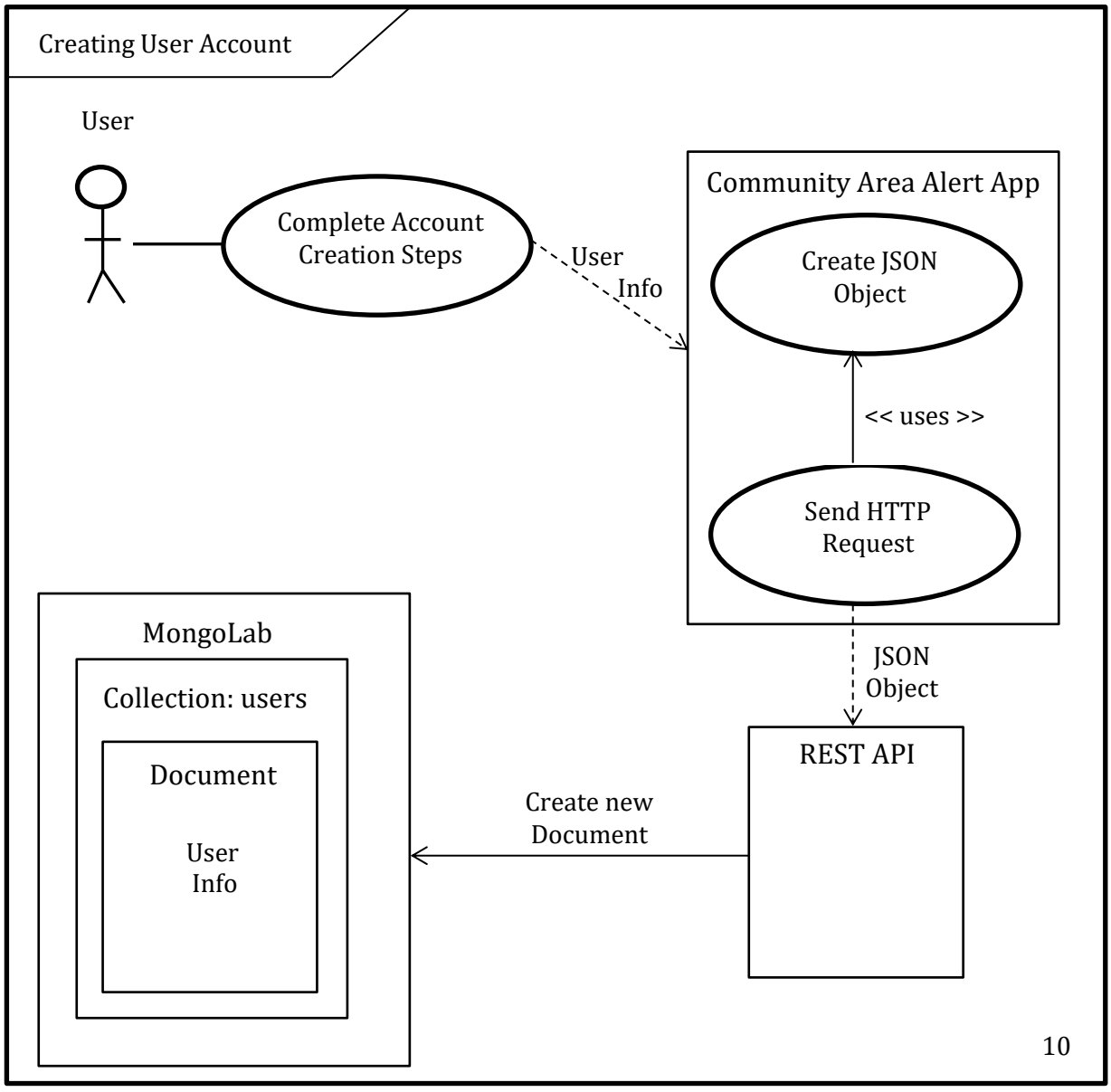
Acceptance testing will be conducted last. Since the Community Area Alert System is not being developed for a particular customer, acceptance testing will be carried out by my family and friends. See the acceptance test plan for more information. A user manual will be produced based on user feedback.

1.1 USER ACCOUNT CREATION

CONNECTING TO MONGOLAB

In order to create a user or admin account on the "CommunityAlert" database hosted on MongoLab I will use the REST API provided by MongoLab.

- This will initially allow me to create a Collection called "users".
- Each user will then have their own document in the "user" collection.
- Each document will have a unique field labelled "_id" which will be set as the "Username" entered by the user during "Step 1" of creating an account.
- Each document in the "user" collection will contain the following fields.
    - "_id" : The users username.
    - "password" : The users password.
    - "dob" : The users date of birth.
    - "lat" : The users latitude coordinate.
    - "lng" : The users longitude coordinate.
    - "verified" : If the users account is verified or not.
- To send the new user account data to MongoLab I will use a HTTPS request with the API key of the "CommunityAlert" database. The information will be sent using a JSON object.



INTEGRATING GOOGLE MAPS

- Google Maps API should be included from the hosted CDN rather than stored locally.
- Google Map should be displayed. It is essential the map contains the option to zoom, scroll and select between satellite and terrain view. These are all options which are provided by the maps API.
- A marker from the Maps API should be added to the map.
- Geo location should be used to get the coordinates of the marker.
- Reverse geo location should be used to get the address from the coordinates.

UI DESIGN PATTERNS

Account Registration

This design pattern will be used to let the user register an account on the Community Area Alert System. This means the system can remember details about the user and in turn use them to personalize what information is displayed to each user.

This design pattern is essential for the Community Area Alert system because:

- The information which should be displayed depends on the user.
- The information presented to each user should be protected.
- Provide personalised content to users.
- I know who is using the system.
- I know the location of users.

Progressive Disclosure

This pattern shall be used to split the account creation process up into 3 steps so that only information relevant to the user's current activity is displayed while other information is delayed.

This design pattern is essential for the Community Area Alert system because:

- De-clutter the interface.
- Only display the next step when the user requests it.
- Only display the next step if the user input is correct so they can focus on each step individually since it's a complex process.

Clear Primary Actions

This will be used to provide simple web forms where the user knows exactly what they are filling out with only one option to submit the form and continue.

This design pattern is essential for the Community Area Alert system because:

- The Community Area Alert account creation process will be split up into 3 steps.
- Fields such as "Username", "Password" or "Date of Birth" will be classed as primary actions. Once each primary action is complete they will only have one option which will take the use to the next step.

Steps Left

This is a useful design pattern when users have to fill in data in multiple steps.

- The user's current position in the account creation process ("eg. Step 2 of3) should be displayed at the top of each page in the process.
- It should be displayed in the same format on each page.

1.2 USER ADDRESS VERIFICATION

CAPTURING IMAGE

- The Cordova Camera plugin should be used with Intel XDK. This will supply camera functions which can allow the user to capture an image.
- Once the user selects to capture an image of a document to verify their address the Cordova Camera Plugin functions should be used to pass control of the Community Area Alert Application application over to the native camera application on the device.
- This ensures an image can be captured on Apple IOS, Google Android and Microsoft Windows Phone.

Example function from the Cordova Camera plugin:

- intel.xdk.camera.takePicture(10,true,"png");
- This function can be used open the native camera application on the device.

STORING IMAGE

- Once the user captures an image it should be saved to the devices local storage. This is essential so that the image can be accessed once control is give back to the Community Area Alert application from the Cordova Camera plugin. Rather than store the image in a particular folder it should be saved directly into the application storage assigned to the Community Area Alert application. The Cordova Camera plugin provides a function to for this.
- Intel.xdk.camera.picture.add
- Once the image is saved it can be accessed using the absolute URL which can be set in the Cordova Camera plugins call back function. This will ensure the Community Area Alert application can access the image when it gets control back after the image is taken.

GENERATING EMAIL

- Generate an email using the user mail client on their device.
- Automatically send email to communityalert@gmail.com by auto filling the "Recipient" field of the email client on the device.
- In order to pre fill the email with the users exact account details the devices mail client can be called using the following function.
- window.open('mailto:test@example.com?subject=subject&body=body');
- The image captured by the user can then be attached to the email using the absolute URL which was set in the Cordova Camera plugins call back function.

GENERATING EMAIL (UPDATED)

Upon implementation, the previous Generating Email design has been deemed to not meet the functional requirements of User Address Verification set out in section 3 of the Functional Requirements document. In order to have a pure Javascript solution using the devices mail client an additional application must be opened which requires user input to send the email and is not automatic.

Mandrill will now be used to generate automatic validation emails.

- Import JQuery from CDN:
  SRC = ajax.googleapis.com/ajax/libs/jquery/1.10.2/jquery.min.js
- Use the Ajax POST function and send the email data to Mandrill at the URL:
  https://mandrillapp.com/api/1.0/messages/send.json
- To send the data all of the required information should be stored into a JSON object.
- The JSON object should include the following keys with specific user values mapped to each.
  - “key” : The api key of my the Community Area Alert Mandrill account.
  - “email” : communityalert@gmail.com
  - “type” : “to”
  - “subject” : “User Verification”
  - “html” : This will act as the body of the email and should include the users username, their address which was generated using reverse geolocation when their account was created and the longitude and latitude of the users address.
  - “images” : This should contain an additional JSON object containing 3 fields.
    - "type": "image/png"
    - "name": "IMAGECID.png"
    - "content": The value of the key “content” should take an BASE64 encoded representation of the image which was captured by the user.
- Encoding an image to BASE64

  Due to security issues an images cannot be taken directly from the storage in a device and encoded. In order to encode an image it must be first copied onto a html canvas. The canvas will have to match the exact dimensions of the image so quality is not lost. Once the image is copied onto a canvas it can be encoded to BASE 64 using the canvas.toDataURL() function.

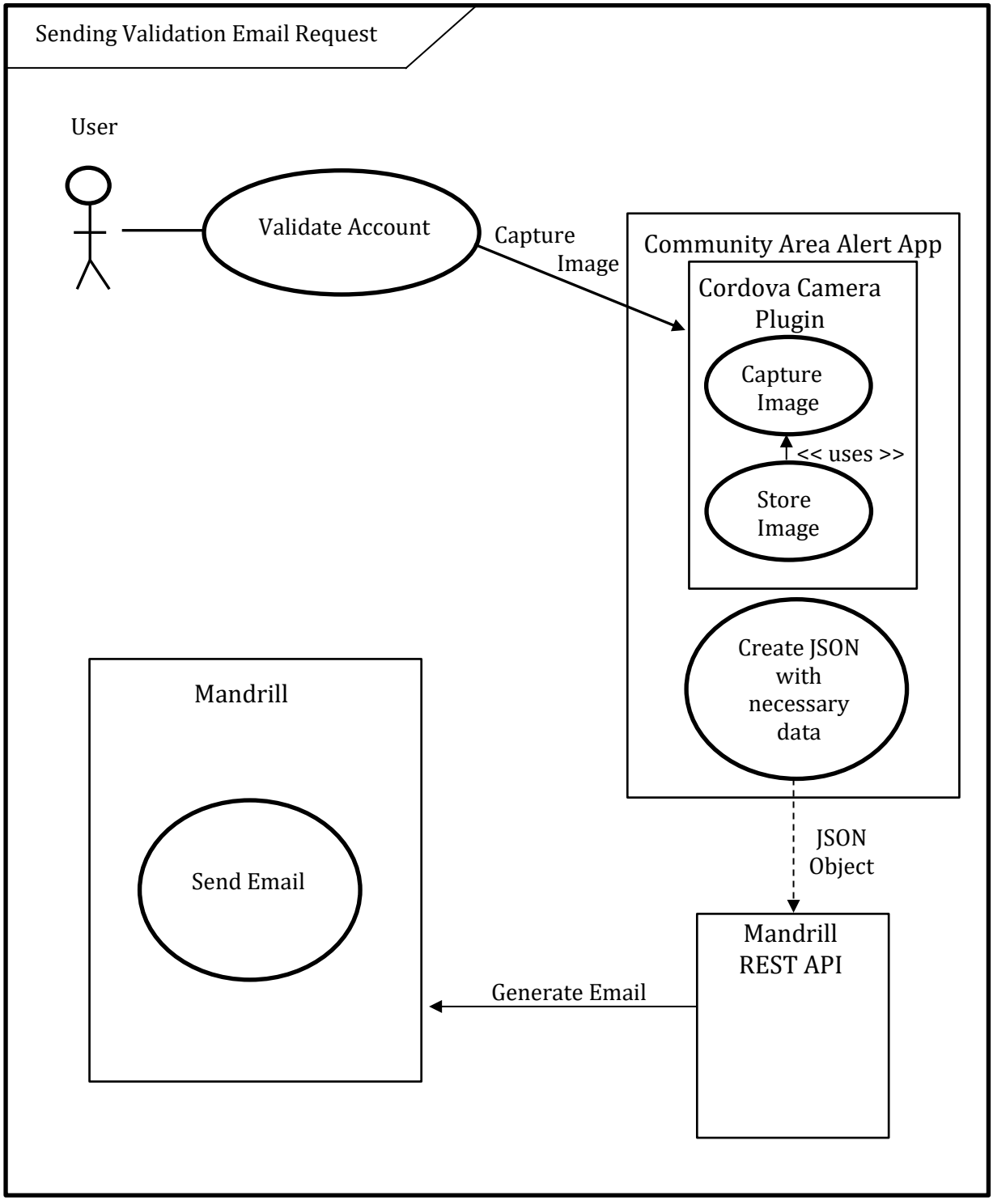
UPDATING USERS VERFICATION STATUS

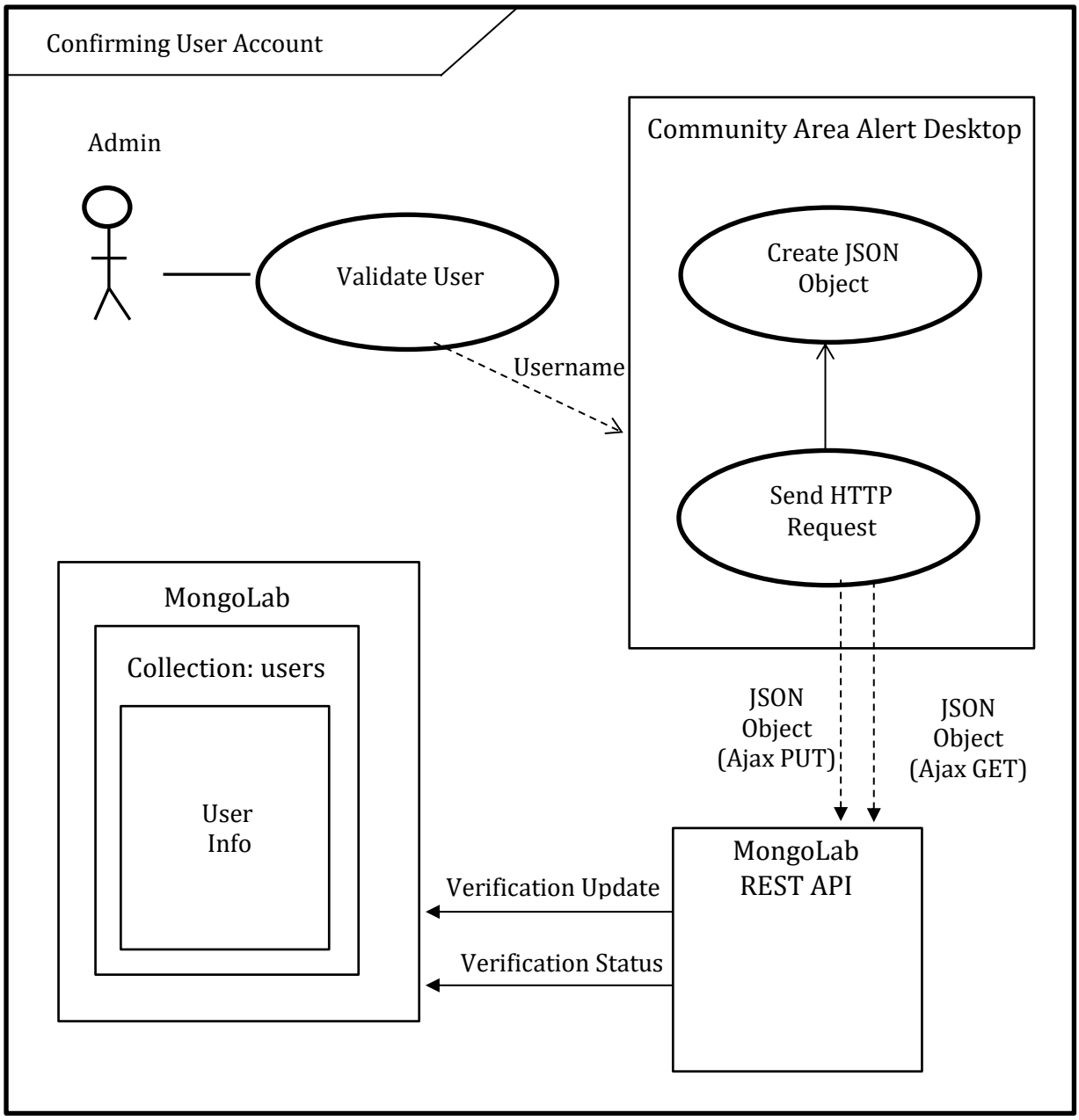
The “Verify User” option on the Community Area Alert desktop application will then be used by an admin who has reviewed the verification request in the email. Updating the user’s verification status to “true” will involve:

- Using JQuery to send an Ajax “GET” request.
- The URL of the request should contain the user’s username as the “document” parameter.

https://api.mongolab.com/api/1/databases/communityalert/collections/users/username?apiKey=MyApiKey

- This GET request will then return a document containing the users account information.
- If the value of the key “verified” is set to “false” the users account can be verified.
- JQuery can then be used to send an Ajax PUT request using the same URL as the users get request. A JSON object containing the key “verified” and the value “true” should be send under the “data” parameter of the PUT request.




Confirming User Account
Admin
Validate User
Community Area Alert Desktop
Create JSON Object
Username
Send HTTP Request
MongoLab
Collection: users
User Info
JSON Object (Ajax PUT)
JSON Object (Ajax GET)
MongoLab REST API
Verification Update
Verification Status

1.3 SIGN IN

CONNECTING TO MONGOLAB

- For a user to sing in to their account the Community Area Alert app must verify there is a "user" document stored on the MongoDB database which matches the inputted username and password.

- For an admin to sing in to their account the Community Area Alert desktop app must verify there is an admin account stored in on the MongoDB database which matches the inputted username and password. Admin account data will be stored in a collection called "stations" as these will represent Garda stations.

- JQuery will be used to send an Ajax GET request.

- The URL of the GET request should contain the inputted username as the "document" parameter.

https://api.mongolab.com/api/1/databases/communityalert/collections/users/username?apiKey=MyAPIKey

https://api.mongolab.com/api/1/databases/communityalert/collections/stations/username?apiKey= MyAPIKey

- If a "404 Not Found" error is returned the username is not valid so the user/admin cannot sign in.

VERIFY INPUTTED VALUES

- If the username entered by a user is contained in a document in the "users" collection in the MongoDB database a JSON object containing the users account information will be returned.

- If the username entered by an admin is contained in a document in the "stations" collection in the MongoDB database a JSON object containing the admins account information will be returned.

- The value of the "password" key in the returned JSON should be checked to ensure it equals the inputted password. If passwords also match the user/admin will be deemed to have satisfied the sign in requirements.

DESIGN PATTERNS

Clear Primary Actions

The user only has one option and knows exactly that their final action to filling out the form will be to click the "Sign In" button and attempt to sign in.

Required Field Marker

The user interface is simplified by only having 2 required fields so the user only has to enter information which is necessary for the sign in interaction. “A rule of thumbs: the simpler and shorter a web form is, the better is the user experience”.

Live Inline Validation

Here the length of the username and password is validated separately as the user types. The colour of the border on each input field will change from red to green to indicate the username or password is within the required length.

- The username of user signing into the mobile application must have a minimum of 4 characters and amaximum of 12 characters. They must have a password with a minimum of 4 and a maximum of 15 characters.
- The username of an admin signing into the desktop application must have a minimum of 4 and a maximum of 20 characters. An admin must have a password with a minimum of 4 and a maximum of 15 characters.
- Since there is not specific standard for username and password length the extra characters allow a Garda station to enter the full name of their station and the extra characters allowed in an admin password may lead from having a longer username.

PSEUDO CODE

```
Get inputted username
Get inputted password

If inputted username and inputted password have a value
Compute Ajax request URL by adding inputted username

Call Ajax GET request


If data is returned from GET request
        If returned password is equal to inputted password
                Sign In user/admin
        Else
                Print “Unknown username or password”

Else if Ajax response is “404 Not Found”
        Print “Unknown username or password”
```

1.4 CREATE NEW REPORT

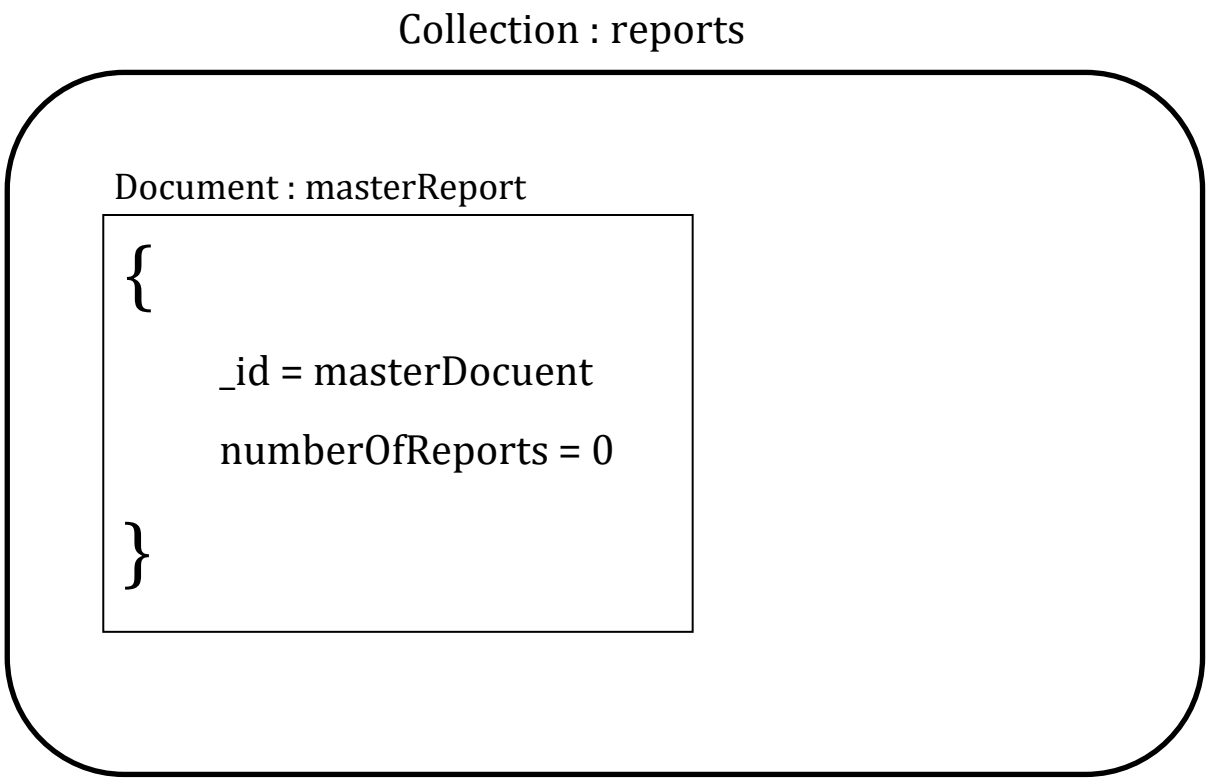
TRACKING REPORTS

In order to create and add new reports to the database I followed the approach of having a "master" document [2] in the database. This master document is responsible for storing wider information on the existence of items in the database.

In my case I shall:

- Store all reports in a collection called "reports".
- Each new report will be added as a new document to the "reports" collection.
- Have an individual document in the "reports" collection called "masterReport". This master report will keep track of the number of reports which have been added to the database under an entry called "numberOfReports".

An example mock-up of the initial state of the reports collection:



STORING NEW REPORTS

Each report should then be added to the "reports" collection as a document containing the following entries:

"_id":

This will be a unique identifier for each report. It will be set to the current "numberOfReports" incremented by one. This will allow the database to scale in size as new reports are added. It will also provide an effective approach for each user to track reports in the database which will

be discussed in section 1.5. The key value "_id" is the key name defined my MongoLab's to identify each document.

"title":

This will represent the title of a report which is entered by an admin while creating the report.

"description":

This will represent the description of a report which is entered by an admin while creating the report.

"lat":

This is the latitude of the report which is taken from the position of the marker place on the map in the location of the report by the admin.

"lng":

This is the longitude of the report which is taken from the position of the marker place on the map in the location of the report by the admin.

"dateTime":

This will represent the time and date that the report was created on. It will be stored in the format dd/mm/yyyy hh:mm:ss. This time and date will be calculated using date/time functions in javascript.

"stationName":

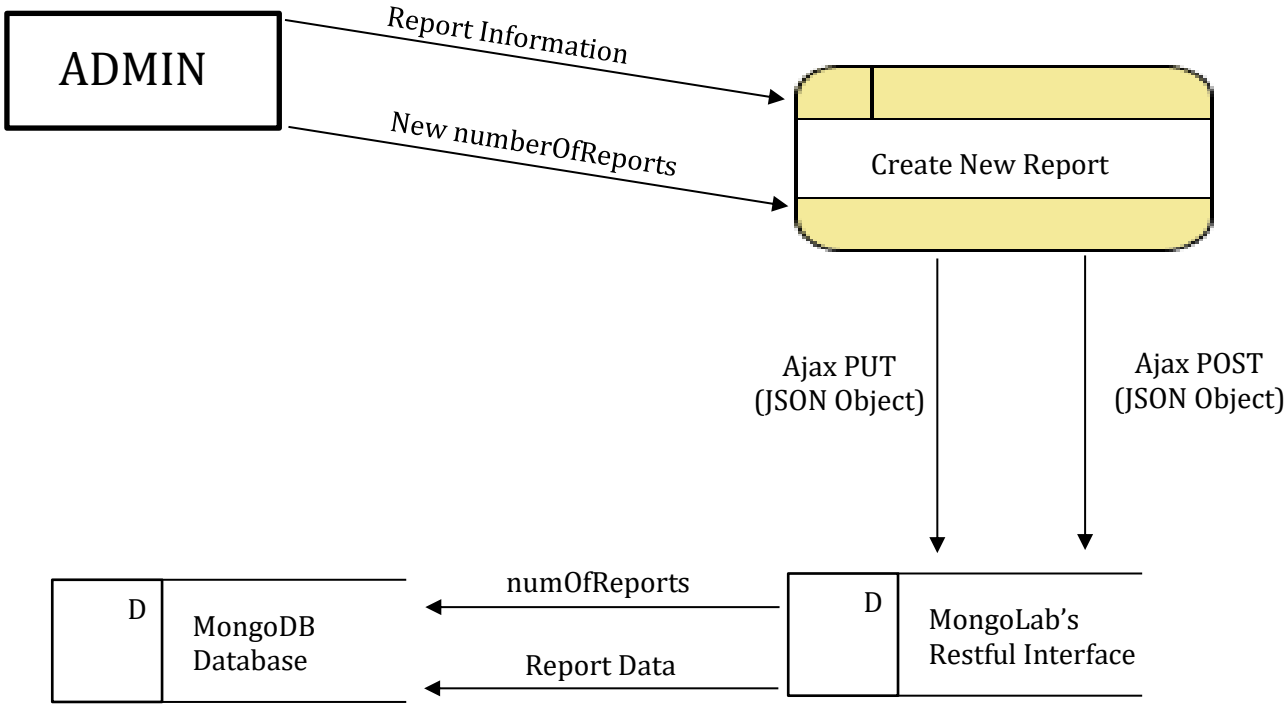
This will be automatically set to the username of the admin account which created the report. An example would be "Kells Garda Station". It is necessary to store this so it can be displayed to the user.

"stationPhoneNum":

This will be automatically set to the phone number of the admin account which created the report. It is necessary to store this so it can be displayed to the user and they know which number to call if they have information on a report.

When the user has entered all the previous data a report can be created. Each report will be stored in a JSON object which will be sent via JQuery using an Ajax POST request to MongoLab's restful interface and stored in the "reports" collection.

DATAFLOW DIAGRAM



STEPS INVOLVED

(i). Get report information entered by user.

(ii). Get the current "numberOfReports" from the "masterReport".

(iii). Set the "_id" of the new report to the "numberOfReports" incremented by 1.

(iv). Create JSON object containing all of the new report information.

(v). Create Ajax PUT request to create a document with the information from the JSON object.

(vi). Increase the number of reports in the materReport to reflect the addition of the new report using an Ajax PUT request to just update the single numberOfReports entry.

1.5 NOTIFICATION

SHORT POLLING

Once each user signs up they should be assigned a report tracking number. This report will be set equal to the current number of reports which is defined under the entry "numberOfReports" in the "masterReport" document contained in the "reports" collection. This means each user will not be concerned with any previously created reports.

- Once a user signs into the mobile application it should poll the database every 15 seconds and check to see if any new reports have been added.
- This technique is called short polling. It involves making a recursive method call containing an Ajax GET request every 15 seconds.
- In my case rather than pulling down the all of the users account information I can simply make an Ajax GET request using JQuery to the "masterReport" using the url: https://api.mongolab.com/api/1/databases/communityalert/collections/reports/masterReport?apiKey=MyAPIKey

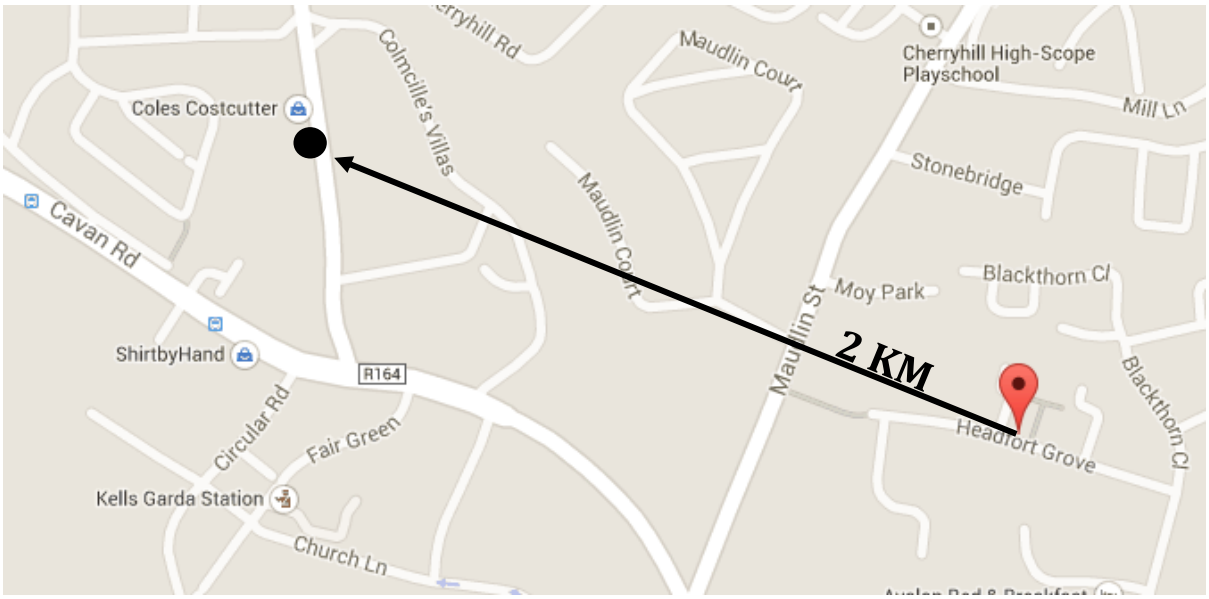
NEW REPORTS

- Short polling every 15 seconds will return the number of reports which have been added.
- I can then compare that number to the users report tracking number. If the users report tracking number "reportTracker" is less than the number of reports taken from the master report the user should download each new report.
- Each time a new report is downloaded it should be checked to see if it is relevant to the user.

CHECKING FOR RELEVANT REPORTS

<u>GPS Radius</u>

If the GPS location of a user's house is within the user's defined GPS radius from the location of the report, then the report is deemed relevant.



In the previous image if a user in the Headford Grove estate has their GPS radius set as 2km or more a report about an incident in the Costcutter carpark would be deemed as relevant to that user.

In order to calculate the distance between the user and the new report I will use haversine formula.

$$\operatorname{haversin}\left(\frac{d}{r}\right) = \operatorname{haversin}(\phi_2 - \phi_1) + \cos(\phi_1)\cos(\phi_2)\operatorname{haversin}(\lambda_2 - \lambda_1)$$

Where haversin is the formula:

$$\operatorname{haversin}(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \frac{1 - \cos(\theta)}{2}$$

This formula can be used to find the distance between 2 points using their latitude and longitude coordinates. The formula can return the distance "as the crow flies" which is exactly what I need.

Each user will have an entry in their user account document called "reports", this will track the id's of all reports which are relevant to that user in the form 1,2,3.

SEDNING NOTIFICATION

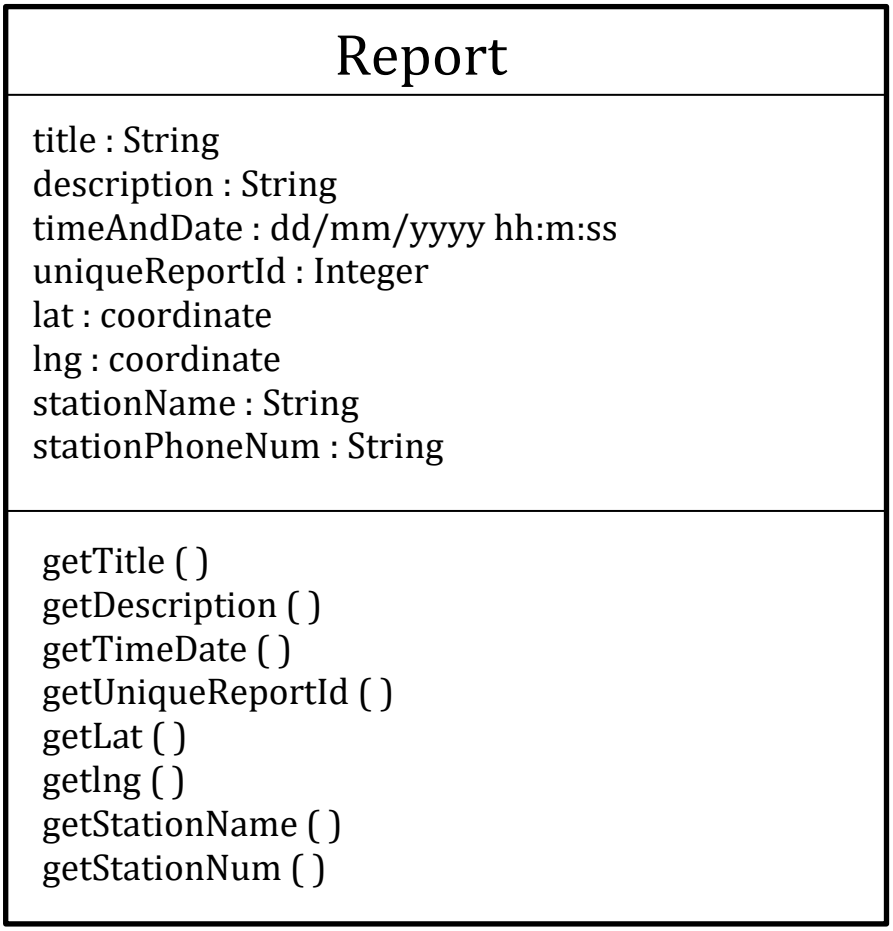
Once one or more new reports are deemed as relevant a notification should be sent to the user.

To notify the user the Community Area Alert application will integrate each user with pushMobi. PushMobi, this will allow me to send Push notifications to a device which the Community Area Alert app is installed on. Each time a user creates an account they be automatically registered as a user on pushMobi. Once the user is registered on pushMobi they will receive a unique id which can be used to tell pushMobi to send a push notification to that user's device. By using push notifications which are sent from pushMobi the device will receive the notification even when the Community Area Alert is not open which is essential.

REPORT CLASS

Each time a short poll discovers 1 or more new reports it should get the details of each of them at once. This creates the need to store each report before it can be checked to see if it is relevant to the user. In order to do this:

- When an Ajax GET request returns a JSON object containing the information of a report, each value from the JSON object should be stored into a Report object.
- Each Report object should be stored into an array of Report objects.
- Once the last report is retrieved from the MongoDB database each Report stored in the array should be check to see if it is relevant to the user.



1.6 VIEW REPORTS

- As previously mentioned each user document will contain an entry called “reports” which keeps track of the reports which are relevant to that user.

- Since the Community Area Alert app will be short polling for updates every 15 seconds this “reports” entry in the user document will always be kept up to date.

PSEUDO CODE

Set users username in document parameter of Ajax URL

Call Ajax GET request for document entry “reports”

```
If data is returned from GET request
        For each report id returned
                Call Ajax GET request for that report id
                Print Display the time, date and title of the report

Else Print “No reports found”
```

Progressive Disclosure

When displaying each report to the user initially only the time, date and title of each report should be shown. All of the other information on a report should be hidden and not displayed unless it is requested by the user. This is a UI design pattern which de-clutters the user interface and puts the user in charge of what they want to view.

1.7 EDIT PREFERANCES

Each user will have the ability to modify the following entries in their user account document.

gpsRadius

This entry will have an initial value of 1 in the user's document. Using the preferences page on the Community Area Alert app a user can update this to a value between 0 and 5 with optional .25 intervals. To avoid user errors when entering numbers a seek bar will be implemented to ensure only a certain type of number is set as the gpsRaduis on the users document.

Examples:

0, 0.25, 0.5, 0.75, 1, 1.25 …….. 4.75 ,5

notifications

This value will have a value of either "true" or "false" on the MongoDB database. To avoid tying errors a switch will be displayed which the user can use to automatically change the value.

Examples:

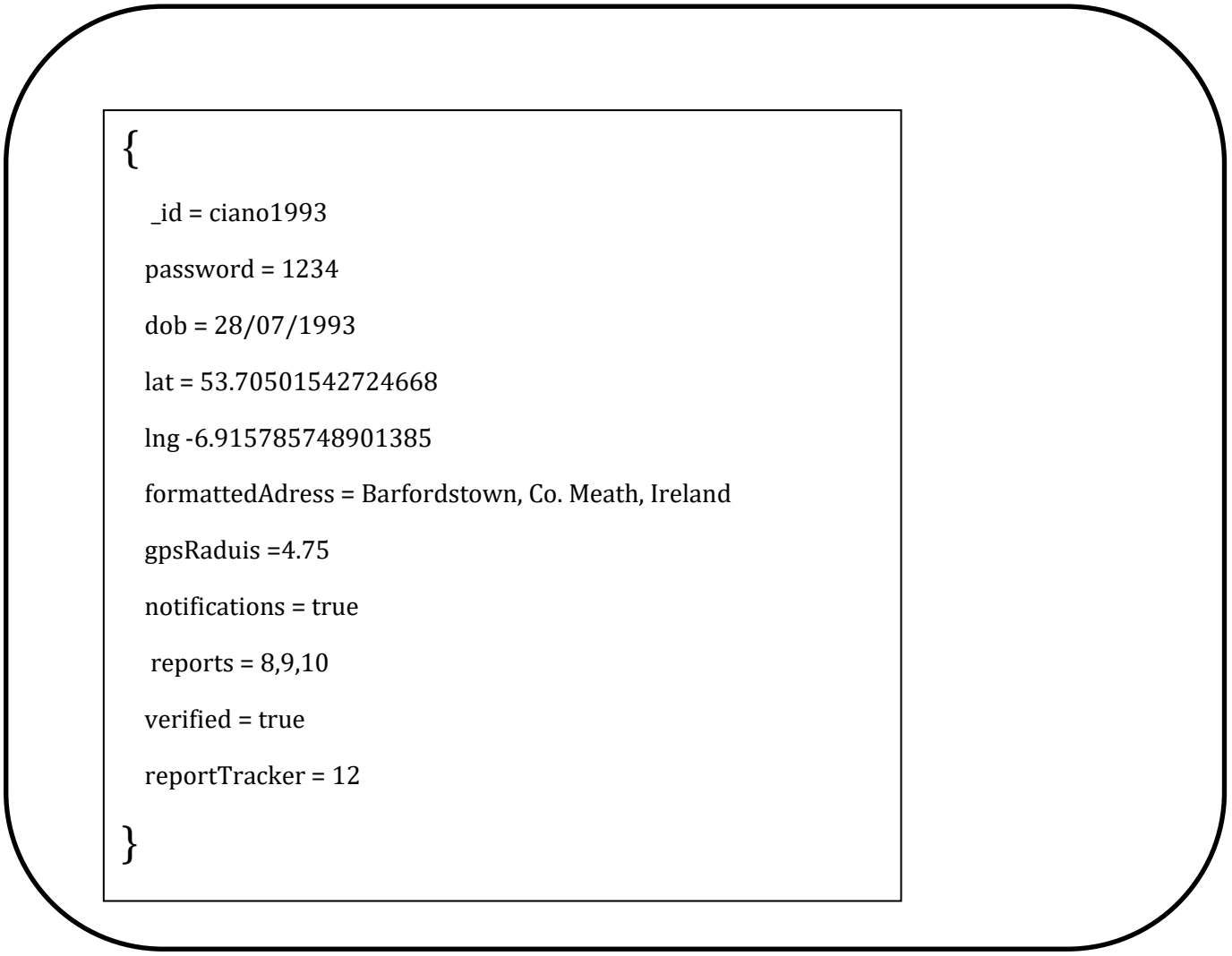
"true" : The user will receive a notification once a poll discovers a new relevant report.

"false" : The Community Area Alert app will still continue to short poll and update the entry "reports" in the users document but it will no longer notify the user when a new report is added.

Each time a user updates a value an Ajax Put request should be sent using JQuery to update the specific entry in the user's document.

The keys “gpsRadius” and “notifications” are the final entries which will make up each user document which stores their account information. An example mock-up of a document containing a user’s account information stored in the user’s collection would look like:

Collection : users



1.8 UPDATE REPORT

The design of this functional requirement involves incorporating many other functional requirements so that a report can be updated.

View Reports

As previously mentioned each admin account should be stored as a document in the collection called “stations”. Each document in the “stations” collection will contain an entry called “reports”, similar to the “reports” entry in each user account. “reports” will store the unique id of each report which was created by that particular station. When an admin attempts to update a report Progressive Disclosure will be used to display the title of each report. The admin can then select a report based on its title. This shall display the description of a report in a text box which can be edited.

Delete Report

Once the admin edits/updates the report description he can select to “Update Report”, this will cause the original report which he updated to be deleted. Each of the steps involved in deleting a report are outline in section 1.9.

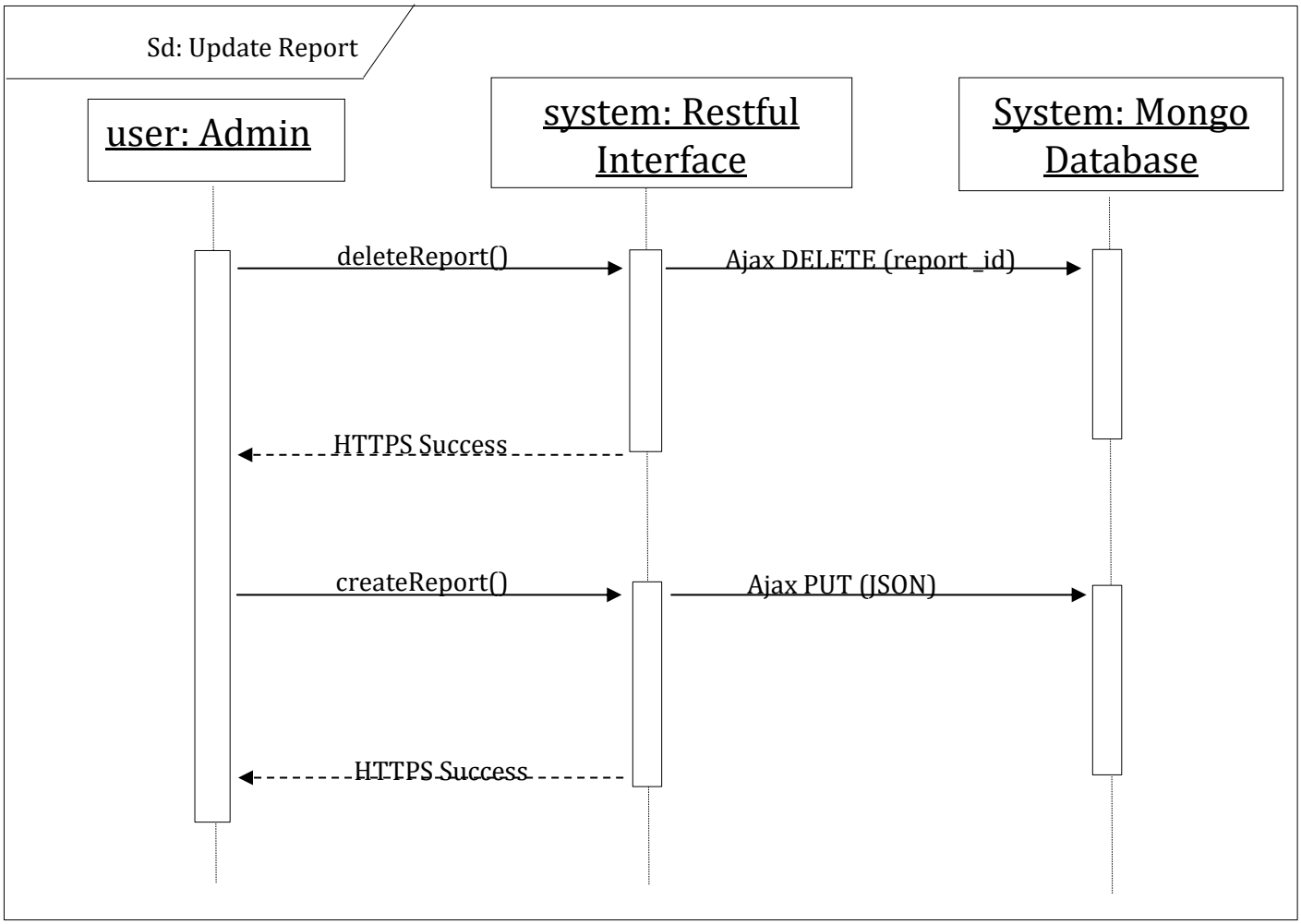
Create New Report

Once the original report is deleted a new report can report can be created. This report will be created using the exact same process involved in section 1.4 and will not be different in anyway even though it is an update to a previous report. This new report will contain all of the updates.

The advantages of this approach are:

- When a user polls the database the updated report will be picked up like a new report and does not require any additional overhead like searching back through reports and checking for edited reports.

- Once the user receives a notification indicating they have a new alert they can refresh their list of reports. A report with the original title will be displayed with a new time and date and an updated description, when the original report is not found it will be removed from that users list of relevant reports. The user will not realise it is actually a new report and it will just appear to be updated.



The final design of a document in the “stations” collection containing information on an admin account should like the following mock-up:

Collection : stations


```
{
   "_id": "Kells Garda Station",
   "password": "1234",
   "phoneNum": "9293423",
   "lat": 53.72858319967284,
   "lng": -6.8818397116089045,
   "reports": 5,10,11
}
```


1.9 DELETE REPORT

Deleting a report can be carried out by an admin who is signed in to the Community Area Alert desktop application. It first involves viewing all reports (1.6) which the particular station has created. The title of each report found in the “reports” entry in the admins account will be displayed. Progressive Disclosure will then be used if the admin wishes to view extra information.

Beside the title of each report an option will be displayed to delete that particular report, once a user selects to delete a report the following steps should be taken to ensure a stable database and no future errors.

- An Ajax DELETE request containing the _id of the report to delete should be sent to the “reports” collection to remove the document containing information on that report.

https://api.mongolab.com/api/1/databases/communityalert/collections/reports/_id?apiKey=MyApiKey

- The “reports” entry in the admins document should be updated to remove the unique id of the deleted report.

- The next time a user refreshes their list of reports the unique _id of the deleted report will still be in the “reports” entry on the user’s document. When a request is made for to get a JSON object of the deleted report a “Not Found 404” error will be returned.
- Once an error is returned it indicates the document has been removed. The “reports” entry in the user’s document can then be updated to remove that unique report id.

CODING STANDARD

During the Code stage of the development process I will follow a coding standard. This will make my code easier to understand and ensures that anyone who looks at the code will know what to expect throughout the entire application. This will also improve the quality of the system overall and make it easier for me to make changes to the system if I notice an issue during the validation phase.

Rather than design my own Coding Standards Document I have decided to use a book written by Robert C. Martin called “Clean Code: A Handbook of Agile Software Craftsmanship”. The reasons for selecting to follow the principals of this book are that they are already widely used in industry but they also follow the same principals which I have been thought throughout my time in college. Some of the main principals you will notice in my code are:

- Meaningful Names
- Small functions with one purpose.
- Keep function arguments to a minimum.
- No comments, explain myself using code.
- Individual source files for each module ("The Newspaper Metaphor”).
- Keep line length under 120 characters.

IMPLEMENTATION

Rather than include all of the HTML, Javacript and CSS files in my project I decided to split this implementation section up into parts based on the functional requirements of the Community Area Alert system. For each functional requirement I will include some code snippets which are important to each functional requirement and talk through them.

1.1 USER ACCOUNT CREATION

Creating an account on the Community Area Alert mobile or desktop application involves 3 steps.

Step 1: Username and Password Input

Most of the code in this step involves checking that the inputted values are the correct length and that the passwords match. Here is an example of 2 functions which check if the username and password are the correct length.

```
function validateUsername()
{
    username = document.getElementById("username_input").value;
    if(!(isEmpty(username))) {
        if(!(isUsernameLengthCorrect(username))) {
            updateUserErrorMessage("Username must be contain minimum 4 charachters and a max of 12.");
        }
    }
    else {
        updateUserErrorMessage("You must enter a username.");
    }
}

function validatePassword()
{
    password = document.getElementById("password_input").value;
    var retypedPassword = document.getElementById("password_input2").value;

    if(isEmpty(password)) {
         updateUserErrorMessage("You must enter a password.");
    }
    else {
        if(isPasswordLengthCorrect(password)) {
            if((compareStrings(password, retypedPassword)) != 0) {
                updateUserErrorMessage("Passwords must match.");
            }
        }
        else {
            updateUserErrorMessage("Password must be between 4 and 15 charachters.");
        }
    }
}
```

This shows a good example of how I implemented the coding standard I discussed during the design document. I keep each function short and explain myself through the code rather than comments. For each "if" statement I could have just included a check for the length of each string but instead I call methods which have clearly defined names and purposes.

```
function isUsernameLengthCorrect(enteredUsername)
{
    return (enteredUsername.length > 3 && enteredUsername.length < 13);
}

function isPasswordLengthCorrect(enteredPassword)
{
    return (enteredPassword.length > 3 && enteredPassword.length < 16);
}
```

Step 2 : Date of Birth and Phone Number

Step 2 of signing up on the mobile application involves entering your date of birth, step 2 on the desktop application involves entering the phone number of your Garda station.

In order to indicate to the user if their input is the correct format such as “24/04/2015/” or “046 9293423” I used pattern matching.

HTML Example from mobile application:

```
<h1>Date of Birth <br>(dd/mm/yyyy):</h1>
    <input id="dob_input" type ="text" placeholder="01/01/2000" pattern="\d{1,2}/\d{1,2}/\d{4}">
```

HTML Example from desktop application:

```
<h1>Station Phone Number</h1>
    <input id="number_input" type ="text" placeholder="Enter the phone number of your station."
           pattern="|^\s*\(?\s*\d{1,4}\s*\)?\s*[\d\s]{5,10}\s*$|">
```

If the pattern of the user input matches the date of birth or phone number pattern the colour of the border on the input box will turn green.

```
input:invalid {
    border: 3px solid red;
    font-size:2em;
    width: 98%;
}

input:valid {
    border: 3px solid green;
    font-size:2em;
    width: 98%;
}
```

You will also notice the each CSS file is laid out based on the coding standard. All class names appear at the top of each file followed by all of the id’s. Both the collection of classes and id’s are displayed in alphabetical order. Each group of elements within a class or id block are also laid out in alphabetical order. Since I decided to stick with the coding standard it meant I could not integrate a UI layout tool such as Twitter bootstrap as it generates a lot of unstructured extra CSS code.

Step 3 : Address Selection

Step 3 on both the mobile and desktop application are similar and involve selecting the location of your address or station using Google maps.

Since this involved using the Google Maps API it was more about integrating the code correctly rather than developing it myself. Here are examples of creating a map and a marker. Both of these functions are then called in the initialization function.

```
function createMap(lat, lng)
{
    var mapOptions = {
        center: new google.maps.LatLng(lat, lng),
        zoom: 8,
        scrollwheel: false,
        streetViewControl: false,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    };
    map = new google.maps.Map(document.getElementById("map-canvas"), mapOptions);
}
```

```
function createMarker(lat, lng)
{
    marker = new google.maps.Marker({
        position: new google.maps.LatLng(lat, lng),
        map: map,
        draggable:true,
        animation: google.maps.Animation.DROP,
        title: 'Your Location'
    });
    google.maps.event.addListener(marker, 'click', toggleBounce);
}
```

The Google Maps API provides functions to determine the user GPS location. Here is a snippet from the initialization function which sets the marker at the user location if the device has GPS enabled, otherwise the marker is placed at the coordinates of Athlone which is the centre of Ireland.

```
success = function(position)
{
    createMap(position.coords.latitude, position.coords.longitude);
    createMarker(position.coords.latitude, position.coords.longitude);
};
error = function()
{
    createMap(53.423933, -7.94069);
    createMarker(53.423933, -7.94069);
}
```

Eventually when the user drags their marker to the correct position the coordinates of their address can be got by once again calling the Google Maps API.

```
lat = marker.getPosition().lat();
lng = marker.getPosition().lng();
```

Creating User Document in Database

As mentioned in the design document an Ajax POST was sent using JQuery to create a document storing the users account information. Here is an example of creating a user account for the Community Area Alert mobile application.

```
function createUserAccount(currentNumOfReports)
{
    getPreviousUserValues();
    var url = "https://api.mongolab.com/api/1/databases/communityalert/collections/users?apiKey=ieVJfwfcyWKcpXwdIU61AGa8qWkAoE32
    $.when($.ajax(
    {
        url: url,
        data: JSON.stringify( { "_id" : recievedUsername,
                                "password" : recievedPassword,
                               "dob": recievedDateOfBirth,
                               "lat" : lat,
                               "lng" : lng,
                               "formattedAddress" : formattedAddress,
                               "community" : "",
                               "gpsRaduis" : 1,
                               notifications : "true",
                               "reports" : "",
                               "verified" : false,
                              "reportTracker" : currentNumOfReports
                               } ),
        type: "POST",
        contentType: "application/json"
    }))
    .then(nextPage, accountCreationError);
}
```

1.2 USER ADDRESS VERIFICATION

The first unique part of integrating this functional requirement was opening the built in camera on the user's mobile device and storing the image.

```
document.addEventListener("intel.xdk.camera.picture.add",function(event)
{
    var name = event.filename;
    var url = intel.xdk.camera.getPictureURL(name);

    updatePageAfterImageCaptured(url);
});
```

One the image is captured it is displayed to the user using the "updatePageAfterImageCaptured" function.

An email can then be generated by sending an Ajax POST request to the Mandrill Restful API:

```
    $.ajax({
        type: "POST",
        url: "https://mandrillapp.com/api/1.0/messages/send.json",
        data: {
            'key': 'kE0NNP5dNDn70hAfGxB57Q',
            'message':
            {
                'from_email': 'cian.mccann9@mail.dcu.ie',
                'to':
                [
                    {
                        'email': 'cian.mccann9@mail.dcu.ie',
                        'name': 'Admin',
                        'type': 'to'
                    }
                ],
                "images":
                [
                    {
                        "type": "image/png",
                        "name": "IMAGECID.png",
                        "content": data
                    }
                ],
                'autotext': 'true',
                'subject': 'User Verfification',
                'html': test
            }
        }
    }).done(function(response)
    {
        var urlSuffix = "?username="+recievedUsername;
        openNextPage("sign_in.html", urlSuffix);
    });
```

Apart from the standard values 2 variables are included in the POST request which are:

1. test

This variable contains HTML code which will make up the body of the email and contains all of the users account information which should be compared against the information on the document captured by the image.

```
var test = "<h2>User Account Verification</h2>
                <h4>The following user would like to verfiy their account:</h4>"
                 + recievedUsername +
                    "<p> They entered the address: </p>"
                      + recievedAddress +
                         "<p>At the coordinates of: </p>"
                            + recievedLat + " , " + recievedLng; + "<img src='cid:IMAGECID.png'>";
```

2. data

This is a Base 64 encoded representation of an image, exactly why this has to be used is described in the design document, however here is the function which copies the image to a HTML canvas and encodes it to Base64.

```
function getBase64Image(img) {
    var canvas, ctxs, dataURL, imageRatio, portraitImage;

    canvas = document.createElement("canvas");

    originalWidth = getOriginalWidthOfImg(img);
    originalHeight = getOriginalHeightOfImg(img);

    var dimensions = scaleToCanvasSize(500, 500, originalWidth, originalHeight);

    canvas.width = dimensions[0];
    canvas.height = dimensions[1];

    ctx = canvas.getContext("2d");
    ctx.drawImage(img, 0, 0, dimensions[0], dimensions[1]);

    dataURL = canvas.toDataURL("image/png");
    return dataURL.replace(/^data:image\/(png|jpg);base64,/, "");
}
```

1.3 SIGN IN

This was one of the most basic functional requirements to implement in terms of having unique functionality. I would like to point out how each functional requirement such as “Sign In” is made up of its own set of unique files. An example for this functional requirement are the 3 files “sign_in.html”, “sign_in.js” and “sign_in.css” this is based off the “Newspaper Metaphor” which is part of the coding standard.

Here is a better example of how the coding standard is applied to a CSS file,

Example: sign_in.css

```css
h1 {
    font-family: font-family: Verdana,Arial,Helvetica,sans-serif;
}

#header
{
    background-color: #00B0FF;
    color: white;
    height: 3.25em;
    margin-bottom: 4px;
    text-align: center;
    width: 100%;
}

#header_text
{
    display:inline-block;
    font-size: xx-large;
    font-family: font-family: Verdana,Arial,Helvetica,sans-serif;
    margin-top: .15em;
    vertical-align:middle;
}
```

Once the user has entered a username and password into the sign in input fields:

```html
<h1>Username:</h1>

    <input id="username_input" type ="text" placeholder="Enter Username" pattern=".{4,12}">


<h1>Password:</h1>

    <input id="password_input" type ="password" placeholder="Enter Password" pattern=".{4,12}">
```

An Ajax GET request can be sent which attempts to download the document where the "_id" is equal to the inputted username. The inputted password can then be compared to the actual password:

```
        $.ajax({
            url: "https://api.mongolab.com/api/1/databases/communityalert/collections/users/" + enteredUsername +"?
apiKey=ieVJfwfcyWKcpXwdIU61AGa8qWkAoE32",
                type: 'get',
            })
            .done(function(data) {
                userAccountInformation = JSON.stringify(data,undefined,1);
                var obj = JSON.parse(userAccountInformation);
                var storedPassword = obj['password'];

                if(enteredPassword == storedPassword) {
                    var lat = obj['lat'];
                    var lng = obj['lng'];
                    var gpsRaduis = obj['gpsRaduis'];
                    var notifications = obj['notifications'];
                    var community = obj['community'];
                    openNextPage("AccountHomepage.html", "?username="+enteredUsername
                                + "&lat=" + lat + "&lng=" + lng + "&notifications=" + notifications
                                + "&gpsRaduis=" + gpsRaduis + "&community=" + community);
                }
                else {
                    sweetAlert("Oops...", "Unknown username or wrong password.", "error");
                }
            })
            .fail(function(request,status,error) {
                sweetAlert("Oops...", "Unknown username or wrong password.", "error");
        })
```

1.4 CREATE NEW REPORT

Once again this functional requirement involves integration with the Google Maps API. This is completed similar to Step of section 1.1 of Implementation. Rather than displaying the latitude and longitude of the location of the report, reverse geo-location is used to convert the coordinates into a user readable address using the following function:

```
function reverseGeoLocateLatLng()
{
    var r = $.Deferred();
    var latlng = new google.maps.LatLng(lat, lng);
    geocoder = new google.maps.Geocoder();
    var infowindow = new google.maps.InfoWindow();

    geocoder.geocode({'latLng': latlng},
    function(results, status)
    {
        if (status == google.maps.GeocoderStatus.OK) {
            if (results[1]) {
                formattedAddress = results[1].formatted_address;
            }
            else {
                formattedAddress= "Could not generate address from coordinates.";
            }
        }
        else {
            formattedAddress= "Could not generate address from coordinates.";
        }
    });
    setTimeout(function () {
        r.resolve();
    }, 2500);

    return r;
}
```

In order to assign the report an id the current number of reports should be taken from the master report document using the following Ajax GET request.

```
function getCurrentStateOfReportCollection()
{
        $.when($.ajax({
            url: "https://api.mongolab.com/api/1/databases/communityalert/collections/reports/masterReport?
apiKey=ieVJfwfcyWKcpXwdIU61AGa8qWkAoE32",
                type: 'get',
            })
            .done(function(data) {
                var reportInfo = JSON.stringify(data,undefined,1);
                var obj = JSON.parse(reportInfo);
                numberOfReports = obj['numberOfReports'];
            })
            .fail(function(request,status,error) {
                sweetAlert("Oops...", "Unknown username or wrong password.", "error");
        }))
        .then(getCurrentListOfStationsReports);
}
```

Once an admin has selected the location and entered a title and description they can select to create a report. This will create a new report in the “reports” collection using an Ajax POST request.

```
function createNewReport()
{
    var newNumberOfReports = parseInt(numberOfReports) + 1;
    var url = "https://api.mongolab.com/api/1/databases/communityalert/collections/reports?apiKey=ieVJfwfcyWKcpXwdIU61AGa8qWkAoE32";
    $.when($.ajax(
    {
        url: url,
        data: JSON.stringify( { "_id" : newNumberOfReports,
                                "title" : reportTitle,
                                "reportDescription" : reportDescription,
                               "lat" : lat,
                               "lng" : lng,
                               "dateTime" : dateTime,
                               "stationName" : username,
                               "stationPhoneNum" : phoneNum
                               } ),
        type: "POST",
        contentType: "application/json"
    }))
    .then(updateMaster(newNumberOfReports));
}
```

Once the report is created the master report and the stations list of relevant reports should be updated also using Ajax PUT requests sent using JQuery.

1.5 NOTIFICATION

As mentioned in the design document a short poll is sent every 15 seconds to check if new reports have been added. Here is the recursive function responsible for this short polling. If no new reports are found it will poll again after 15 seconds.

```
function pollForUpdates(usersTrackedNumberOfReports)
{
    newRelevantReports = "";
    $.ajax({
        url: "https://api.mongolab.com/api/1/databases/communityalert/collections/reports/masterReport?
apiKey=ieVJfwfcyWKcpXwdIU61AGa8qWkAoE32",
            type: 'get',
        })
        .done(function(data) {
            var reportInfo = JSON.stringify(data,undefined,1);
            var obj = JSON.parse(reportInfo);
            numberOfReports = obj['numberOfReports'];
            if(usersTrackedNumberOfReports < numberOfReports) {
                getNewReports(usersTrackedNumberOfReports, numberOfReports);
            }
            else {
                setTimeout(function() { pollForUpdates(usersTrackedNumberOfReports); }, 15000);
            }
        })
        .fail(function(request,status,error) {
            sweetAlert("Oops...", "Check your internet connection.", "error");
    })
}
```

If new reports are found each new report will be downloaded and stored into an array of "Report" objects.

```
function Report(title, description, timeAndDate, uniqueReportId,
                lat, lng, stationName, stationPhoneNum)
{
    this.title = title;
    this.description = description;
    this.timeAndDate = timeAndDate;
    this.uniqueReportId = uniqueReportId;
    this.lat = lat;
    this.lng = lng;
    this.stationName = stationName;
    this.stationPhoneNum = stationPhoneNum;

    Report.prototype.getTitle = function()
    {
        return this.title;
    };

    Report.prototype.getDescription = function()
```

The difference in km between the user's location and the location of each report is then checked to determine if the report is relevant to the user.

```
 if (distanceBetweenReportAndUser < gpsRaduis) {
        newRelevantReports = newRelevantReports + "," + possibleReleventReport.getUniqueReportId();
 }
```

As mentioned in the design document Haversine's formula was used which returns the distance between the 2 sets of coordinates in kilometres.

```
function getDistanceFromLatLonInKm(lat1,lon1,lat2,lon2)
{
    var R = 6371;
    var dLat = deg2rad(lat2-lat1);
    var dLon = deg2rad(lon2-lon1);
    var a =
    Math.sin(dLat/2) * Math.sin(dLat/2) +
    Math.cos(deg2rad(lat1)) * Math.cos(deg2rad(lat2)) *
    Math.sin(dLon/2) * Math.sin(dLon/2);

    var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    var d = R * c;
    return d;
}
```

Where r is the radius of the earth in km.

```
function deg2rad(deg)
{
  return deg * (Math.PI/180)
}
```

Any new reports which are found to be relevant are then added to the users "reports" entry in their document in the "users" collection. The users tracking number must also be updated so older reports are not downloaded repeatedly.

```
function updateUserReports(userReports)
{
    var url = "https://api.mongolab.com/api/1/databases/communityalert/collections/users/" + username + "?
apiKey=ieVJfwfcyWKcpXwdIU61AGa8qWkAoE32";
    $.ajax(
    {
        url: url,
        data: JSON.stringify( { "$set" : {"reports" : userReports,
                                          "reportTracker" : numberOfReports} }),
        type: "PUT",
        contentType: "application/json"
    })
    .done(function(data)
    {
        getReportTrackingNumber();
    })
}
```

1.6 VIEW REPORTS

A user or admin can request to view relevant reports which are stored in the “reports” entry of their user account. An Ajax GET request is used to get the value of the “reports” entry related to the user or admin account.

An example of this is when a user of the Community Area Alert mobile application presses the refresh button. Since the number of reports which are relevant to the user at that time is unknown, each report has to be downloaded and displayed on the page dynamically. This involves originally creating a “mainDiv” which will contain each report.

```
<div id="inner_body">
    <div id="mainDiv">

    </div>
</div>
```

For each report which is found a div called “innerDiv” should be dynamically created using javascript. This “innerDiv” will contain the title of each report along with the time and date it was created. This “innerDiv” can then be appended to the “mainDiv” to display it on the page.

```
var mainDiv = document.getElementById("mainDiv");

var innerDiv = document.createElement('div');
innerDiv.className = 'innerDiv';

var uniqueReportID = index.getUniqueReportId();
innerDiv.innerHTML = '<img class="edit_pic" id="' + uniqueReportID
                    + '" onclick="editExpandCollapseImage(this.id)" src="images/expandim.png">';
var time = document.createElement('h3');
var title = document.createElement('h3');
time.innerHTML = index.getTimeAndDate();
title.innerHTML = index.getTitle();
innerDiv.appendChild(time);
innerDiv.appendChild(title);
```

However a user will expect to see more than the title, time and date of each report. A hidden container called “reportContent” will be dynamically appended to each “innerDiv”, this “reportContent” div will contain all other information on a report such as the description and the name and phone number of the station who reported it. Once again the previously mentioned “Report” class will be used to store information on each report.

```
var reportContent = document.createElement('div');
reportContent.className = "content";

var description = document.createElement('h3');
description.innerHTML = index.getDescription();

var stationName = document.createElement('h3');
stationName.innerHTML = index.getStationName();

var stationPhoneNum = document.createElement('h3');
stationPhoneNum.innerHTML = index.getStationPhoneNum();

reportContent.appendChild(description);
reportContent.appendChild(stationName);
reportContent.appendChild(stationPhoneNum);
innerDiv.appendChild(reportContent);
```

Finally based on the Progressive Disclosure design pattern JQuery is used to slowly expand the hidden "reportContent" container when the expand image is selected for a particular report. This expand function has to be assigned dynamically to the expand button of each report.

```
$('img#'+uniqueReportID).click(function()
{
    $header = $(this);
    $content = $header.next().next().next();
    $content.slideToggle(500, function ()
    {
        this.src = "";
    });
});
```

1.7 EDIT PREFERENCES

Rather than using the standard HTML seek bar I implemented an open source seek bar I discovered at http://www.jqueryrain.com/?0qRLJKK8. As mentioned in the design document this is used to allow the user to select a GPS radius which is used to determine if a report is relevant.

The user can adjust the value of the seek bar between 0 and 5 km. A call back function is then used to display the current value of the seek bar to the user.

```
$(function()
{
    $(document).on('input', 'input[type="range"]', function(e)
    {
        document.getElementById("sliderValue").innerHTML = "Alert Radius: " + e.target.value + " km";
    });

    $('input[type=range]').rangeslider({
        polyfill: false
    });
});
```

As mentioned in the design a switch should be used to allow a user to turn notifications on and off to reduce errors. To implement this I generated a switch using https://proto.io/freebies/onoff/.

Based on the value of the "notifications" entry in the users account document, the initial value of the switch can be simply set using the following function. The value of the "notifications" entry is taken using an Ajax GET request,

```
function dispayUserAccountValues()
{
    if(notifications == "true") {
        document.getElementById("myonoffswitch").checked = true;
    }
    else {
        document.getElementById("myonoffswitch").checked = false;
    }
}
```

When the user updates the value of the seek bar or on off switch they can press the save button. This will update the values in their user document using an Ajax PUT request which replaces a specific value of an entry such as “gpsRadius” which was updated by the user.

```
$.ajax(
{
    url: url,
    data: JSON.stringify( { "$set" : {"gpsRaduis" : gpsRaduis} }),
    type: "PUT",
    contentType: "application/json",
    success: displaySettingsUpdatedMessage,
    error: function()
    {
        sweetAlert("Oops...", "Cannott update settings. Check internet connection.", "error");
    }
})
```

1.8 UPDATE REPORT

In order to update a report an admin must first download and view all reports which their station has created. This involves an Ajax GET request to the stations account stored in the “stations” collection. Once the value of the “reports” entry in the stations document is retrieved, each unique report id must be downloaded and stored, once again the Report class is used.

Each report which was downloaded and stored in a Report object should be added to an array of reports. Similar to how a user views reports, the title, time and date of each report is dynamically added to the page by appending a div called “innerDiv” to the “mainDiv”.

```
function displayReportInDiv(index)
{
    var mainDiv = document.getElementById("mainDiv");
    var innerDiv = document.createElement('div');
    innerDiv.className = 'innerDiv';
    var uniqueReportID = index.getUniqueReportId();
    innerDiv.id = uniqueReportID;

    innerDiv.innerHTML = '<img class="edit_pic" id="' + uniqueReportID
                         + '" onclick="setValuesInUpdateFields(this.id)" src="images/edit.png">'+
                         '<img class="edit_pic" id="' + uniqueReportID
                         + '" onclick="confirmAlert(this.id)" src="images/delete.png">';
    var time = document.createElement('h3');
    var title = document.createElement('h3');
    time.innerHTML = index.getTimeAndDate();
    title.innerHTML = index.getTitle();
    innerDiv.appendChild(time);
    innerDiv.appendChild(title);
    mainDiv.appendChild(innerDiv);
}
```

Notice how an image “edit.png” is added to each “innerDiv”, clicking on this edit image will allow the user to update the report. Once you select the “edit.png” image for a particular report the “id” of the image will indicate which report should be displayed to be updated. This is because the “id” of each “edit.png” represents the index of the related report in the array of downloaded reports.

The function “setValuesInUpdateFields” uses Progressive Disclosure to show elements on the page which are then set to the values of the report the user wishes to update. Each element is displayed by setting the visibility=”” as this was previously set to visibility=”hidden”.

```
function setValuesInUpdateFields(uniqueReportID)
{
    var titleInputForm = document.getElementById("report_title");
    titleInputForm.innerHTML = reports[uniqueReportID].getTitle();
    var descriptionInputForm = document.getElementById("report_description");
    descriptionInputForm.value = reports[uniqueReportID].getDescription();
    var heading = document.getElementById("updateHeading");
    var updateButton = document.getElementsByClassName("updateBtn")[0];
    updateButton.id = uniqueReportID;

    titleInputForm.style.visibility = "";
    descriptionInputForm.style.visibility = "";
    heading.style.visibility = "";

    document.getElementsByClassName("updateBtn")[0].style.visibility = "";
}
```

Once the description of the report the user wishes to edit is displayed in the "descriptionInputForm" the user can edit or update the report and then select the "Update Report" button.

Next the original report which was updated can be deleted, the reasoning behind this is displayed in the design document.

```
function deleteOriginalReport()
{
    $.ajax
    (
        {
            url: "https://api.mongolab.com/api/1/databases/communityalert/collections/reports/"+ indexOfReportToUpdate +"?apiKey=ieVJfwfcyWKcpXwdIU61AGa8qWkAoE32",
            type: "DELETE",
            async: true,
            timeout: 300000,
            success: function (data) { updateStationsListofReports() },
            error: function (xhr, status, err) { }
        }
    );
}
```

A new report can then be created (Implementaion 1.4) with the updated description and timestamp.

1.9 DELETE REPORT

In order to delete a report an admin must first download and view all reports which their station has created. This involves an Ajax GET request to the stations account stored in the "stations" collection. Once the value of the "reports" entry in the stations document is retrieved, each unique report id must be downloaded and stored, once again the Report class is used.

Each report which was downloaded and stored in a Report object should be added to an array of reports. Similar to how a user views reports, the title, time and date of each report is dynamically added to the page by appending a div called "innerDiv" to the "mainDiv".

This time we focus on the "delete.png" which is added to the "innerDiv" container of each report. Once again each "delete.png" has an "id" which relates to an index in the array of Reports containing all the information on that report.

```
innerDiv.innerHTML = '<img class="edit_pic" id="' + uniqueReportID
                        + '" onclick="setValuesInUpdateFields(this.id)" src="images/edit.png">'+
                        '<img class="edit_pic" id="' + uniqueReportID
                        + '" onclick="confirmAlert(this.id)" src="images/delete.png">';
```

Once the “delete.png” image is selected the “id” of the image will be the index in the array of reports which contains all the information on that report. One bit of information it contains is the “_id” which is the unique identifier for that report in the “reports” collection in the database. An Ajax DELETE request can then be used to remove that document from the collection.

```
function deleteOriginalReport()
{
    $.ajax
    (
        {
            url: "https://api.mongolab.com/api/1/databases/communityalert/collections/reports/"+ indexOfReportToUpdate +"?
apiKey=ieVJfwfcyWKcpXwdIU61AGa8qWkAoE32",
            type: "DELETE",
            async: true,
            timeout: 300000,
            success: function (data) { updateStationsListofReports() },
            error: function (xhr, status, err) { }
        }
    );
}
```

The “reports” entry in the stations account should then be updated to remove the unique id of the deleted report. Remember a “reports” entry is in the format of “1,2,3”. “reportIds” is the array with each unique report id created by the station.

```
var stationReports;

if(reportIds.length == 1) {
      stationReports = newNumberOfReports;
}
else {
    var indexOfDeletedReport = reportIds.indexOf(indexOfReportToUpdate);
    if (indexOfDeletedReport > -1) {
        reportIds.splice(indexOfDeletedReport, 1);
    }
    reportIds.push(newNumberOfReports);
    stationReports = reportIds.join();
}
```

As this project was also a learning experience in HTML, javascript and CSS a lot of the code had to refactored. In the earlier stages of development I was learning as I was developing so it was hard to stick to a coding standard. Here is an example of how I refactored “verify_address.js”:

```
function processUser()
{
    var parameters = location.search.substring(1).split("&");
    var temp = parameters[0].split("=");
    recievedUsername = unescape(temp[1]);
    temp = parameters[1].split("=");
    recievedPassword = unescape(temp[1]);
    temp = parameters[2].split("=");
    recievedDateOfBirth = unescape(temp[1]);
    temp = parameters[3].split("=");
    recievedLat = unescape(temp[1]);
    temp = parameters[4].split("=");
    recievedLng = unescape(temp[1]);
    temp = parameters[5].split("=");
    recievedAddress = unescape(temp[1]);
}
```

The ”processUser” method is long and hard to follow. The name of the function was not clear, this was mainly due to the function not having clear a simple purpose. This method was refactored into:

```
function getUserValuesFromUrl()
{
    return parameters = location.search.substring(1).split("&");
}

function getUrlValue(index)
{
    var temp = index.split("=");
    return unescape(temp[1]);
}
```

```
function getUserDetails()
{
    var parameters = getUserValuesFromUrl();
    recievedUsername = getUrlValue(parameters[0]);
    recievedPassword = getUrlValue(parameters[1]);
    recievedDateOfBirth = getUrlValue(parameters[2]);
    recievedLat = getUrlValue(parameters[3]);
    recievedLng = getUrlValue(parameters[4]);
    recievedAddress = getUrlValue(parameters[5]);
}
```

These 3 functions are smaller, more readable and each has its own specific purpose.

Another example in the “verify_address.js” is refactoring of the functions involved in capturing an image and displaying the captured image on the page. The original code looked like:

```
function captureCamera()
{
    intel.xdk.camera.takePicture(10,true,"png");
}

document.addEventListener("intel.xdk.camera.picture.add",function(event)
{
    var name = event.filename;

    var url = intel.xdk.camera.getPictureURL(name);
    imgSRC=url;

    document.getElementById("captured_image").style.visibility = "";
    document.getElementById("captured_image").style.height = "150px";

    document.getElementById("uploadBtn").style.visibility = "";
    document.getElementById("uploadBtn").style.width = "";
    document.getElementById("uploadBtn").style.height = "";
    document.getElementById("captureBtn").style.marginRight = "5px";
    document.getElementById("image_btn_container").style.paddingBottom = "5px";
    document.getElementById("captureBtn").value="Retake";

    document.getElementById("captured_image").setAttribute("src", url);
    var capturedImage = document.getElementById("captured_image");

    originalWidth = getOriginalWidthOfImg(img);
    originalHeight = getOriginalHeightOfImg(img);
});
```

Once again this function is long and hard to follow. The name does not clearly describe what the function intends to do. By breaking this function in smaller functions it becomes a lot more understandable.

```
function captureCamera()
{
    intel.xdk.camera.takePicture(10,true,"png");
}

function displayCapturedImageInContainer(container, imgSCR)
{
    container.setAttribute("src", imgSCR);
    container.style.visibility = "";
    container.style.height = "150px";
}

function displayButton(button)
{
    button.style.visibility = "";
    button.style.width = "";
    button.style.height = "";
}

function changeElementText(element, newText)
{
    element.value=newText;
}
```

```
function updatePageAfterImageCaptured(url)
{
    var capturedImageContainer = document.getElementById("captured_image");
    displayCapturedImageInContainer(capturedImageContainer, url);

    var uploadButton = document.getElementById("uploadBtn");
    displayButton(uploadButton);

    var captureImageButton = document.getElementById("captureBtn");
    changeElementText(captureImageButton, "Retake");

    captureImageButton.style.marginRight = "5px";
    document.getElementById("image_btn_container").style.paddingBottom = "5px";
}

document.addEventListener("intel.xdk.camera.picture.add",function(event)
{
    var name = event.filename;
    var url = intel.xdk.camera.getPictureURL(name);

    updatePageAfterImageCaptured(url);
});
```

PROBLEMS SOLVED

One of the biggest issues I had was due to my decision to follow a coding standard. Following the coding standard meant I had to generate all the code myself. This meant I could not use something like Twitter Bootstrap which would have made designing the user interface of the mobile and desktop applications a lot easier. I originally tried to using Twitter Bootstrap but it was generating a lot of extra code which did not follow the coding standard. The only way to really overcome this was to increase my knowledge of HTML and CSS so it took a lot longer for me to design and positon UI elements. In the end I am happier with the result as I feel the Community Area Alert mobile and desktop applications have a more unique look and feel.

I regret not implementing my own server. It would have taken some extra work at the start but it would have saved me time overall. I felt like I could not follow the standard approach when implementing functionality such as sending emails and notifications. To solve this problem I had to use alternative approaches which usually meant using Restful API's.

Since I managed to implement each functional requirement, the Community Area Alert system provides a solution to the main problem with the current Community Area Alert scheme, the costs. Since free notifications can be sent to alert users, this reduces the need for a 15 euro signup fee. In an area where the scheme is working successfully sending free alerts could save up to 1200 euro per year according to James O'Neill. My Community Area Alert system can therefore make the system sustainable, as James said "the success of the scheme may be its own downfall" due to cost.

Another problem with the current scheme is the complexity involved in setting up a Community Area Alert scheme and getting members signed up. By allowing users to enter their location anyone who is signed up the Community Area Alert mobile application can receive alerts based on their GPS location. This reduces the complexity of setting up and running a scheme but should still bring the all the benefits.

It can also provide An Garda Siochana with an alternative to posting alerts on Facebook, they can now send alerts directly to users and choose who they want to see a particular report. This reduces the need for members of the pubic to share posts from the Garda Facebook and spread information. It also means the An Garda Siochana have full control of the alerts which are sent out. They do not require a member of each community to keep a special phone with them. They can directly send alerts through the Community Area Alert desktop application and reduces the reliance on the public.

RESULTS

I feel like I have completed my goal and designed a cost free alternative to the current Community Area Alert scheme. The current scheme is having a very positive effect on areas where it has been implemented successfully. I feel like my system can perform everything that the current scheme is required to do, as well as fixing some of its issues and improving the overall efficiency of the process. I am happy with how the system looks and functions at the moment but it will require some more work I mention in the "Future Work" section before I contact the Munitir Na Tire Director of Services James O'Neill to give a demo of the system. I still think like the chances of the system being implemented by an Garda Siochana are low but I feel like the work I completed for my fourth year project has definitively given me an opportunity to present the system in a positive an professional manner.

I also feel like I have achieved my goal of using a lot of the information which I learned in the last four years of college. I managed to implement validation, software process, user interface design principals, design patterns, pattern matching, database design, systems analysis, UML, web design, object oriented design principals etc. There are aspects of my project which involve something from every semester of my course. I also feel this project has helped me gain a greater skill set in terms of programming languages. Although we did have a module on web design in first year I intended to use this project to learn HTML, javascript and CSS. I now feel like I can develop in these languages to a high enough standard that I could place them on my CV. I feel like I not only gained a strong knowledge of each language but I also developed an interest in learning and increasing my skills in web development even further. I feel like using HTML5 to develop mobile applications is a better approach to using the Android Development Kit which I used for my third year project.

I felt like the biggest obstacle in developing this Community Area Alert system was the schedule. I had some other ideas I will mention in the future work section which were not included in the scope of this project. Once I fell behind it was impossible to catch up, the area of the project which suffered because of this was the cross platform integration of the Community Area Alert mobile application. At the moment the app meets all of its functional requirements across Andorid, iOS and Windows mobile operating systems but due to schedule I did not get to tailor the interface for each device so the app is not ready for production on iOS or Windows.

In terms of what I would do differently there is nothing major I would change about my approach. Using a design pattern might not have been the best idea but I wanted to maintain what I had learned in intra and I made a commitment to do this project to industry standard which meant a design pattern was necessary. I think if I was able to go back in time and start the project again I would create a Node.js server. I had originally intended to do so but changed my mind before creating the Functional Specification when I realised it was not necessary. However I felt like I was always looking an alternative way to do things such as using Mandrill to send an email rather than php or using pushMobi to send notifications. I do not feel like not having my own server has reduced the quality of the Community Area Alert system but I do feel that if I had taken a few days at the start to get my own server up and running that it would have saved me some time in the long run.

FUTURE WORK

- Some refactoring will be required before the project demonstration.
- I will have to plan how I intend to present the Community Area Alert system before the project expo.
- I intend to give a demonstration to my project supervisor to see if he can give me any pointers on what to concentrate on for the project expo and demonstration.
- I intend to design and develop some additional functional requirements for the system which were outside the scope of my fourth year project. I intend to implement a PayPal so that user of the mobile application can pay for a bundle of text messages. By using an SMS API I can then send text alerts to users who have paid for text alerts and notifications to users who have no not paid for text alerts or have no text alerts remaining on their account.
- I intend to contact James O'Neill from Muintir Na Tire, he said he was interested in seeing the system and could get me an opportunity to present it in front of An Garda Siochana.
- If it gets to the stage where the system might be implemented I would have fix the display on iOS and Windows OS, as I mentioned before it currently does not display 100% on these devices but all of the functionality is in place.