Software Design Study

'Final Report'

UNIVERSITY OF BIRMINGHAM

Emergency Systems Team

George
Matthew Flint - 1247903 - mxf203@bham.ac.uk
Deyan
Martin
Robert

This work was conducted as part of the requirements of the Module 06-15500 Software Design Study of the Computer Science department at the University of Birmingham, UK, and is submitted in accordance with the regulations of the University's code of conduct.

April 30, 2015

Contents

1 Abstract

The software design study gives the student the opportunity to work in a team (typically 5 or 6 students) on a challenging and substantial software design project. This will normally include mainly the early phases of the software lifecycle (requirements analysis and software design), but may also include the further development of prototype software and/or demonstration software.

2 Introduction

3 System Overview

4 Modules

4.1 Emergency App

4.1.1 Video Streaming Rationale

Video calling is not a recent development, but one that has been processed throughout time [33]. It has finally emerged and is making quite the splash in the technology world. Mobile phones with video calling capabilities provide instant face-to-face communication with anyone, anywhere. In order to have this capability, the phone must have a camera and a stable internet connection.

Nowadays, most mobile phones have built-in cameras [34][35] with the capability of capturing images with a resolution of at least 2 MP, which is of HD quality or transferred into pixels: 1600x1200. Some mobile phones provide even more than 41 MP resolution of a single image. These cameras, however, record video with a lower quality than single images. Older phones with older 2 MP camera sensors can record video with resolution of 240p, which is equal to 320x240, at 15 frames per second. With the advances of technology, this has changed a lot. Nowadays, most mobile phones can record videos with at least 480p (640x480) resolution and at least 30 frames per second. The newest models from 2014 even have the possibility of recording video with 4K resolution (4096x2160) at 30 frames per second. Apart from high quality of the video, modern mobile phones can exchange quality for speed and capture video with 720p (1280x720) at 120 fps or some even reach 240 fps.

Another feature that is required in order to make a video call and all mobile phones with built-in cameras have is mobile or wireless internet connection. Mobile connection is usually slower, starting with 2G, 3G and reaching the newest and the most advanced 4G. As of June 2014, the 2G and 3G coverage of the UK premises is 99.7% and 99.5%, respectively [36]. Nevertheless, further technology improvements are being developed. 4G coverage has already reached 73.0% of UK premises and researchers has began work on 5G, which is believed to be introduced in the early 2020s [37].

Based on statistics by Skype [38], a normal internet call requires a minimum of 30 kbps of download and upload bandwidth. In order to be able to conduct a simple video call using a

Technology	**Coverage**	**Speed**
2G (GPRS)	99.7%	56 to 115 kbps
2G (EDGE)	99%	up to 237 kbps
3G	97%	0.2 to 28 Mbit/s
4G	73%	100 Mbps

smartphone, a bandwidth of minimum 128 kbps upload and download speed is needed, whereas 300 kbps is recommended. For higher quality of the call the speed requirements rise to 400kbps and 1.2 Mbps for the highest quality. Youtube suggests similar network requirements for its streaming service [39]. Taking into account the use of newer encoding techniques, the network requirements can be compared with the network speed that mobile technology provides. Using 2G, a simple video call can be made, with low quality settings. However, 3G speeds are enough in order to perform a medium to high quality stream and 4G will be able to provide even better streams.

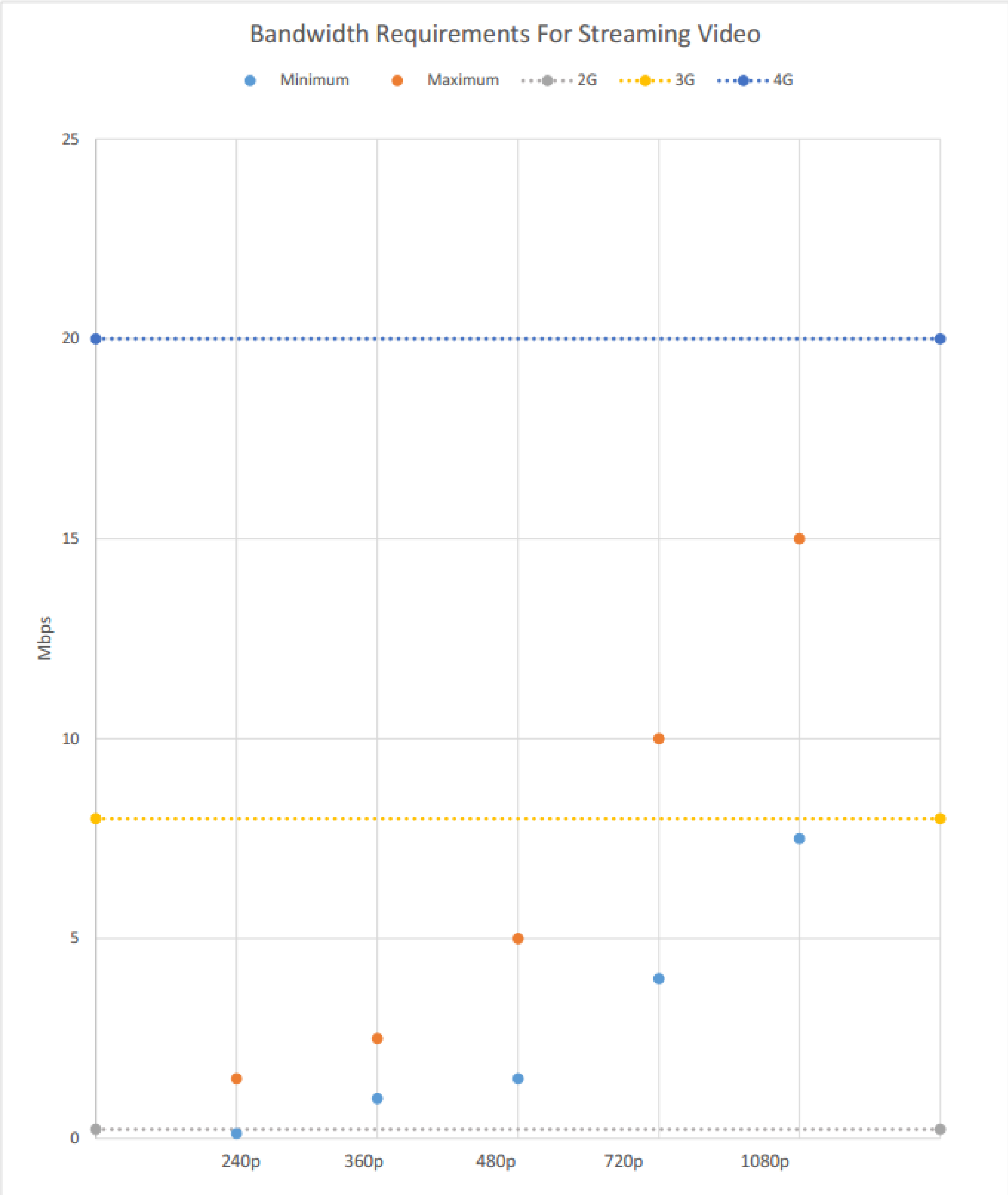


Figure 1: The bandwidth requirements for video streaming.

Wireless internet connection can be compared to the same network requirements. As it can usually provide better and more reliable internet connection, it would be the prefered way of transferring the video stream data. However, free wireless internet connection is available in limited places, thus, a combination between wireless and mobile internet connect is required, in order to provide internet access of the highest quality and coverage.

Thanks to the low bandwidth requirements and fast mobile network that is available almost everywhere, it is now possible to make video calls using your mobile phone wherever you are. Furthermore, with the technology evolving so fast, it is likely to be able to make High Quality video calls from everywhere in the near future.

These technology innovations could change how emergency situations are handled. Being able to send real time video footage to an emergency operator or an emergency team, could help them assess a given situation better and give better instructions to the people requesting help. In a phone conversation we held with a personnel of an emergency department, we asked to express what are her thoughts on introducing video streaming feature to an emergency call. That would be very useful. One of the issues we always have, when we take a 999 call, is that we are blind. We are going on what our caller is telling us, so to be able to see it, would make things a lot easier and we could assess the severity a lot quicker. - Dawn Whelan. This was also confirmed by a survey that we have conducted, which showed that 82% of the people, who participated, think that they could benefit from such a feature.

4.1.2 Chat System Rationale

As of 2012, there are more than 32,000 registered phones with the Emergency SMS service [52]. However, according to statistics [53] there are more than 10 million people with some kind of hearing loss and 800,000 of them are severely or profoundly deaf. That means that only about 4% of the people, who might experience the need to use the Emergency SMS service, are registered. Including a live chat feature in the emergency application will greatly reduce the difficulty of using such a service and will make it more accessible to a greater part of the population.

Using a live chat requires a minimum amount of internet access, which means that the chat will be available for use even in regions where mobile network coverage is poor. Moreover, such chat will be able to deliver messages in the same order they were sent, keep track whether a message has been delivered and read by the operator and will make requesting more information easier and faster. A single message will take only milliseconds to be delivered, rather than the slow speed of delivering SMS messages. If a message fails to be delivered, the application can keep trying or notify the user that it was unable to send the message. Giving such information to the users, rather than suggesting to them to resend the message if no response has been received in 3 minutes, gives them the opportunity to react faster.

4.1.3 Automatic Video Sending Rationale

A problem that may arise during a 999 call is that the caller may not be able to describe in details where he is located. This could be due to him being in an unknown place or just confused because of stress. The results of such a problem can be that the emergency team head in the wrong direction or arrive to the place of the emergency, but are unable to find the exact location of the accident. In order to help with transferring the exact location of the person, requesting emergency, mobile phones can be used.

As already stated, nowadays, all mobile phones have the ability to access internet, through various technologies. Apart from that, most mobile phones have GPS sensors [34][35] and are able to locate themselves within seconds. The acquired location can either be exact, within a few meters, if a GPS sensor is available and a good GPS signal is present, or at least identifying the correct region, based on mobile network reception. This information is also very small, in terms of data size, which means that it can easily and quickly be sent over the network. Including such a feature in the emergency application can greatly reduce the time needed for the emergency team to head in the right direction, thus, the team will be able to arrive faster.

4.2 CPR System

4.2.1 Rationale

Cardiovascular diseases are one of the leading causes of death in the western world [1]. They come with an increased risk of cardiac arrest, of which there are an estimated 60,000 [2] incidents out of hospital annually in the UK - about 1 every 9 minutes. In fact, across the whole of Europe, 1 person per 1000 population will suffer a cardiac arrest in any year.

The truth however, is that in most cases of out of hospital cardiac arrest the chances of survival are depressingly low. The average overall survival rate for England is just 8.6% [2] and in some parts of the country, just 1 in 14 people survive an unanticipated cardiac arrest [3]. This is poor by international standards, with some of the highest survival rates being Norway (25%), Holland (21%) and Seattle (20%) [2] which shows clear potential for improvement in the UK.

The most effective way to increase a persons chances of survival from cardiac arrest is to perform immediate CPR, whether that be only chest compressions or mouth to mouth. Evidence suggests that where CPR is attempted, survival rates are doubled [5] and this could be expected to save around 300 lives per year. This is because the chance of survival after a cardiac arrest reduces by around 10% every minute without proper care [6] due to the lack of oxygen the body (and especially the brain) experiences. Early CPR until paramedics arrive is very important to maintain blood circulation to the heart and brain, which also increases the chance that treatment with defibrillators is successful [6].

One of the main reasons that the fatality rates of the UK are so high is because of a low rate of initial CPR by bystanders: Fewer than one in five people who suffer a survivable cardiac arrest receive the life-saving intervention they need from people nearby [3]. Compare this to 73% in Norway [4] and it is clear that this is an area for major improvement. There are several factors thought to be responsible for low levels of bystander-initiated CPR, including lack of training and fear of litigation [5]. In addition, the number of population trained in CPR is currently 3.8m [5] (out of 60m). This small proportion means the likelihood of someone being nearby when an individual suffers a cardiac arrest is very low, and furthermore, training new people to give CPR will only help in the long term as it will take some time before any difference to this likelihood is noticed. In the meantime, a better way is needed to link those trained to give CPR to those in need of it.

So few recoverable cardiac arrests are survived mainly due to the time it takes between the arrest occurring and the rescue attempt beginning. With the average waiting time currently around 8 minutes [7], and recent news of longer response times from emergency services [24], we need to look into ways of helping those affected before emergency services arrive.

4.2.2 Project SMS-livrddare

There is an ongoing research project in Sweden called Project SMS-livrddare [6], which aims to improve cardiac arrest survival rates by getting trained civilians to start CPR early, before the ambulance arrives [6]. Currently active in the entire capital city of Stockholm, the trial started in May 2010 and has seen massive uptake from the public, with 9,600 residents currently registered [9]. This has resulted in SMS-livrddare-volunteers reaching victims before ambulances in 54% of cases and has helped increase survival rates from 3% to nearly 11%, over the last decade [8].

The system works by having willing civilians trained in CPR register to help if a cardiac arrest happens in their vicinity. When an emergency call is received, the geographical position of the caller is determined. If there is suspicion that a cardiac arrest has occurred the emergency operator activates a positioning system that locates the mobile phones of helpers connected to the service. In cases where a lifesaver is nearby, they are alerted via their mobile phone. Meanwhile ambulance and emergency services are alerted. The alarm to the SMS-lifesavers mobile phone comes as an SMS. The text message contains information from the emergency services about where the suspected cardiac arrest has occurred and the message also includes a map link which can be used to more easily find the location. The SMS-lifesaver also receives an automatically generated phone call to alert the user that an SMS arrived on the phone [6].

This is the basis for our implementation of this feature. We believe that this project has done incredible work and shown people a new way of being a part of first aid assistance, but that there are areas which could be refined and improved. For example, the project currently has users register two areas that they will likely be, one for day and one for evening, then uses these areas to determine people to contact. This, and other problems like it, will be what we intend to address in our implementation of the system.

4.3 API

---

5 Specification

This section contains the technical specifications of the modules and how we have designed them throughout the iterations.

5.1 Common Technologies

5.1.1 Google Cloud Messaging

5.1.1.1 Design Overview For our system we intend to utilize push notifications as the main way of initialising contact between our server and the users phone. As our system is initially being developed for the Android platform, we are going to implement the Google Cloud Messaging (GCM) [10] system as the main method of sending and receiving these notifications.

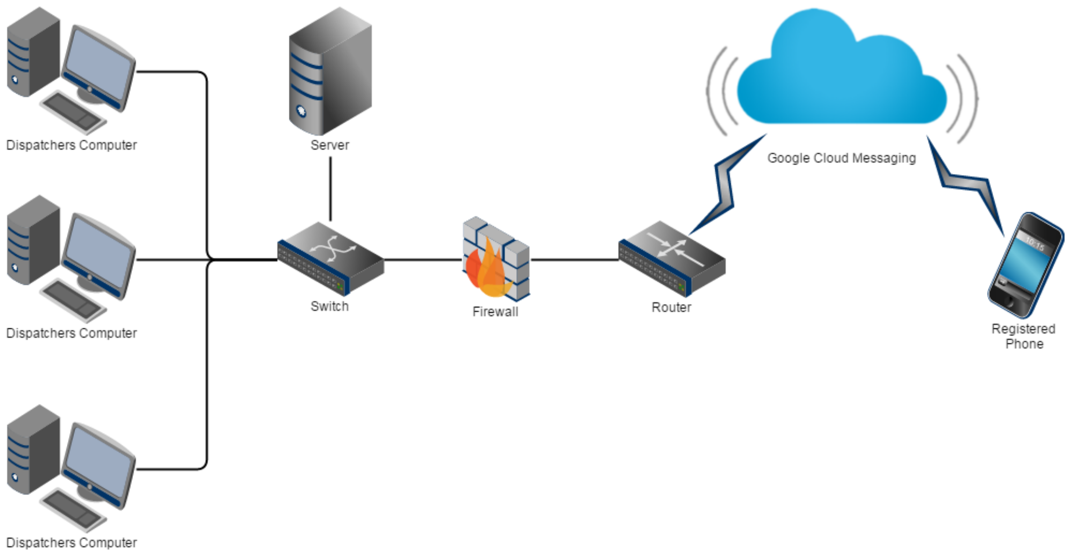


Figure 2: Network map of how our system will communicate between devices and the server

We have chosen to use this system for a few reasons. Firstly as our app is initially being developed for Android (due to current limitations with IOS) so using GCM is advisable as it is heavily embedded with the operating system and so reduces strain (such as power usage) [11]. GCM also allows the notifications to be received when the app is not running which is of huge benefit to the user who then does not have to keep the app open to offer their help. Other notable reasons include that it is free to use, can handle large scale push notifications [11], notifications can be canceled at a later time using a collapse key and that you can send data (up to 4096 bytes) as a payload to be used by the app. GCM can also be integrated with ISO push notifications [13] which could be useful if, in the future, an IOS app was being developed.

As a result of deciding to use GCM, we need to keep a database of the registered app users. This is because GCM uses a registration ID system to send a message to a specific phone, we need to store and relate this ID to a user of our system.

5.1.1.2 GCM Setup We intend to use GCM primarily for its ability to push notifications to a clients device. There are several things which we need to have set up before the notifications can be sent. Firstly, each phone will have a unique key associated with it which is used to identify an individual's phone so messages can be sent to it. This is obtained by the app when it registers with the GCM service. In order for our server to send messages to this device we need to know and store this registration ID, therefore the device will need to send it to our server to be stored in a secure database. The detailed setup instructions are available directly from google and have example code to assist in the setup [14].

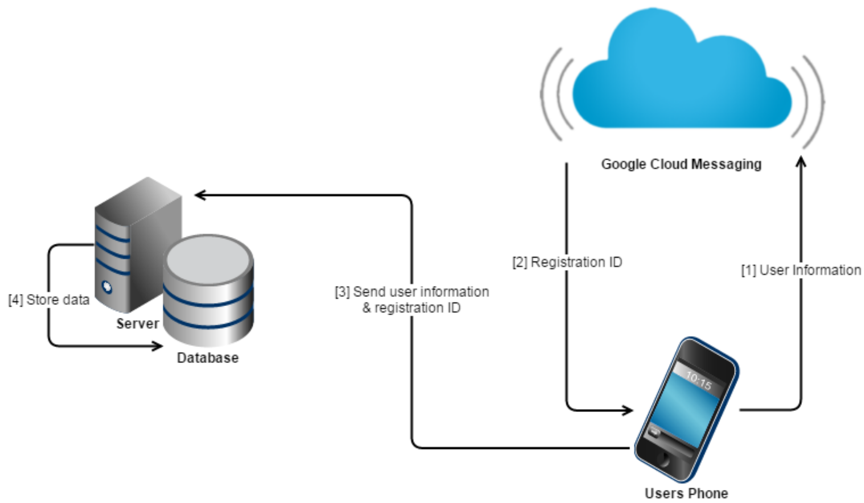


Figure 3: Diagram of the initial setup and registration with Google Cloud Messaging

We will also have to setup the database to store the information of the user that the phone is registered to along with the Registration ID (token) obtained from GCM. This design is detailed in the database section.

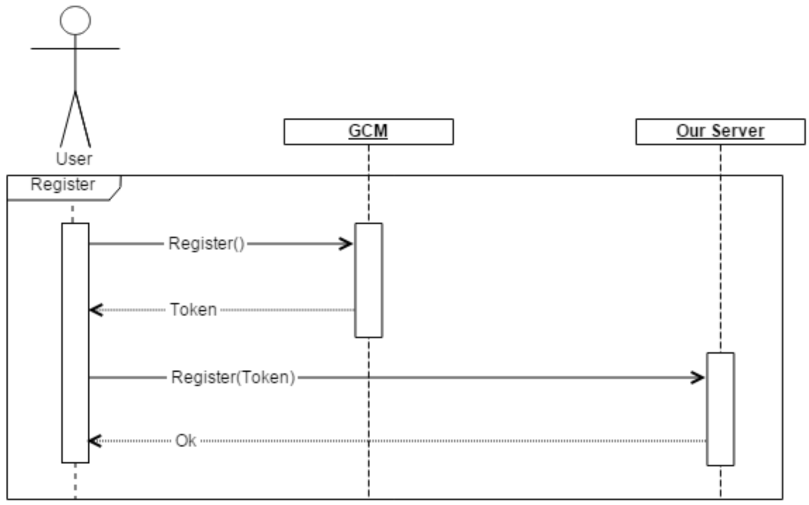


Figure 4: Sequence diagram showing interactions when registering a device

5.1.1.3 Sending Downstream Messages Once the initial setup is completed we can focus more on the message sending and receiving parts of the application. In order to push a notification to our users phone our server needs to send a request containing the registration ID and the message to Googles Messaging Server which then sends it directly to the device associated with that ID.

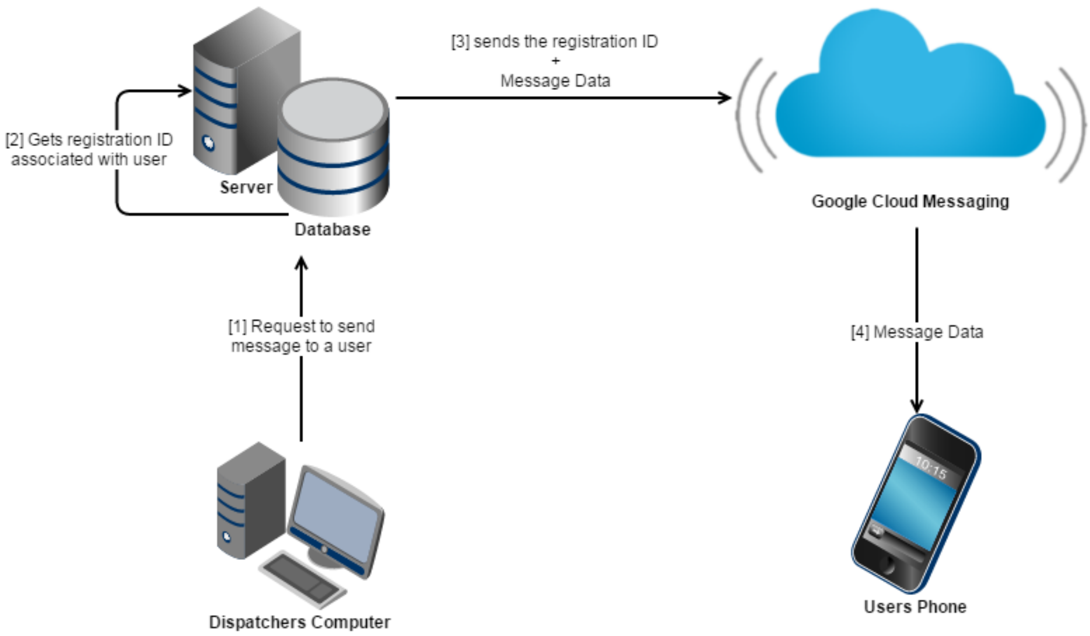


Figure 5: Diagram illustrating sending of a message to a clients device

We have decided to use the XMPP protocol as the connection between our server and GCMs server. This was chosen (over HTTP) for a few reasons. Firstly, the XMPP protocol is faster than HTTP and so would allow the system to send messages to users quickly which is a crucial aspect of any emergency system. It also uses the existing GCM connection on the device to receive the data, saving battery useage as you don't have to open your own connection to the server. XMPP does not however allow broadcast messages to be sent to multiple devices, instead you have to send a new message for each device you wish to contact. This should not present a problem in our system as we will only need to send a message to one device (in this iteration) and even if a later iteration was to require sending to multiple devices, it would be a small number.

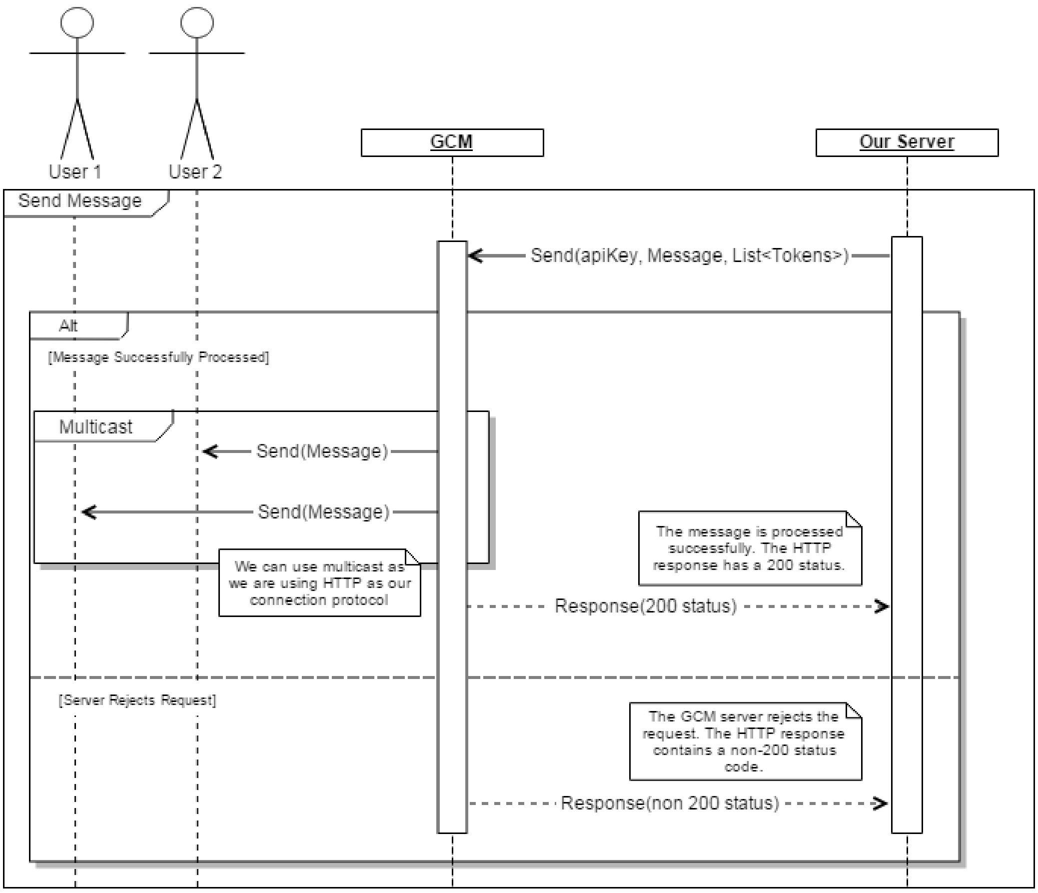


Figure 6: Sequence diagram showing the interactions when sending a message to the client device

Once a XMPP connection is established our server can use normal XMPP ¡message¿ stanzas to send a JSON-encoded message. For this iteration our message will contain five components, the registration ID of the device we are sending the message to, a message ID to uniquely identify this message, a collapse key we can use to overwrite the message later, a time-to-live after which the message with expire and not be sent, and the payload data we wish to send. The format of the message will look like this :

```
<message id="">
  <gcm xmlns="google:mobile:data">
  {
      "to":"REGISTRATION_ID",
      "message_id":"UNIQUE_MESSAGE_ID",
      "collapse_key":"UNIQUE_COLLAPSE_KEY"
      "time_to_live":"TIME_IN_SECONDS",
      "data":
      {
          "KEY":"VALUE",
      }
  }
  </gcm>
</message>
```

For each device message your app server receives from GCM, it needs to send an ACK message. If you don't send an ACK for a message, GCM will just resend it. GCM also sends an ACK or NACK for each message sent to the server. If you do not receive either, it means that the

TCP connection was closed in the middle of the operation and your server needs to resend the messages[10].

These response messages will need to be dealt with according to their responses. The ACK will be sent if the message was successfully delivered however there are two types of error messages (NACK and Stanza error) for situations such as Invalid JSON, bad registration ID or Device Message Rate Exceeded. Each of these messages contains an error ID and message which informs you of what the problem is so that it can be dealt with [15].

In some situations, we will include a payload of data that will be sent (in the message) to the clients device. This payload is easily incorporated into the message and forms part of the JSON-encoded message string. This payload data has no limit to the number of key-value pairs however there is a total limit of 4kb maximum message size. String values are recommended to all other data types would need to be converted to strings before sending them to the device [15].

5.1.1.4 Sending Upstream Messages We can also use GCM to send upstream messages from the clients device to the server via Googles CCM (Cloud Connection Service). This works by embedding the servers registration ID (obtained when registering for the GCM service) into the clients software and passing this ID along with a message to the GCM service to be forwarded to the server.

Figure 7: Diagram showing the sending of a message from the client device to the server

This will once again use the XMPP protocol for the reasons stated in Sending Downstream Messages and then we can use normal XMPP ¡message¿ stanzas to send a JSON-encoded message in the following format[25] :

```
<message id="">
  <gcm xmlns="google:mobile:data">
  {
      "category":"com.example.yourapp", // to know which app sent it
      "data":
      {
          "KEY":"VALUE"
      },
      "message_id":"UNIQUE_MESSAGE_ID",
      "from":"DEVICE_REGISTRATION_ID"
  }
  </gcm>
</message>
```

This message is then forwarded by GCM to our server and the message is parsed for its data. The device uses the send() method from the GCM API to construct and send the message. This takes the following format:

```
gcm.send(GCM_SENDER_ID + "@gcm.googleapis.com", id, ttl, data);
```

Where GCM_SENDER_ID is the stored GCM ID number of the server, id is the unique message ID, ttl is the time to live of the message (seconds) after which the message will expire and not be sent and finally the data to be sent [25]. This will produce a stanza in the format described above and send it to the server ID specified.

After the message is received from CSS by our server we are expected to send an ACK message back to google. This should be in the following format [25]:

```
<message id="">
  <gcm xmlns="google:mobile:data">
  {
      "to":"DEVICE_REGISTRATION_ID",
      "message_id":"MESSAGE_ID_OF_RECIEVED_MESSAGE"
      "message_type":"ack"
  }
  </gcm>
</message>
```

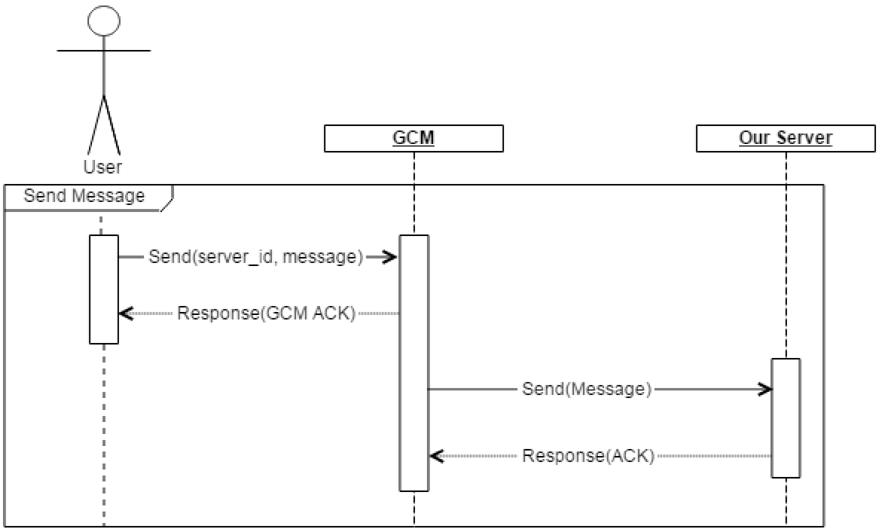


Figure 8: Sequence diagram showing the interactions when sending a message to the server from a client device

5.1.2 Database

It was immediately apparent that both the Emergency App and CPR System would need a database backend. To ensure we chose the best system, we investigated the three most prolific database management systems (DBMSs): MySQL, Oracle, and SQL Server.

Almost immediately, we discounted MySQL as being immature compared to Oracle and SQL Server. Transactions and stored procedures are a relatively recent addition to the system, and both are technologies that we will rely on heavily to ensure data is consistent and intact. It also has far fewer features relevant in enterprise environments, lacking fine-grained access control, table partitioning and disaster recovery.

Next, we scrutinised the remaining contenders. Ultimately, there isnt a lot of difference between them; SQL Server has a marginal advantage with in-memory tables and better performance in write-intensive environments, while Oracle is supposedly more suited to applications with equal read and write traffic due to its locking system.

In the end, we decided to go for SQL Server due to its Spatial Data extensions, which would be used particularly heavily in the CPR System. This functionality allows us to do accurate geometric calculations, like finding the distance between two sets of coordinates, within the database itself. In addition, SQL Server is renowned for its stability, with a transaction and journaling system that detects and automatically fixes errors, preventing corruption and increasing uptime - all of which are desirable traits in such life-critical applications.

5.2 Emergency App - Video Streaming

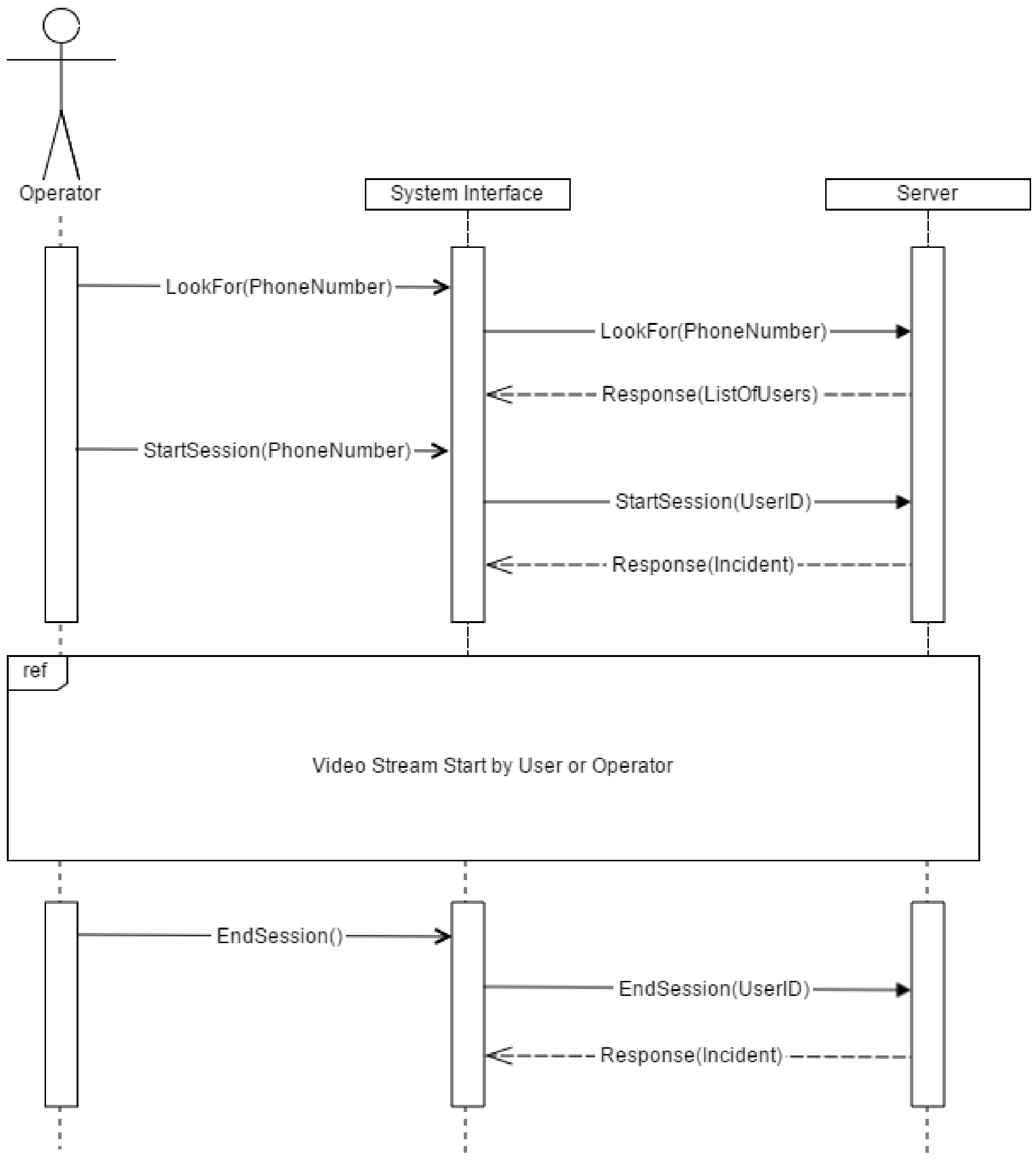
5.2.1 Iteration 1 - Basic Functionality

5.2.1.1 Aims This iteration develops the core functionality of the video stream service. It is intended to provide the most basic features of making a video stream, with all the security operations implemented.

At the most basic level, the user has to be able to start and stop the video stream and the operator has to be able to request the start and stop of the video stream and see it on his computer screen. Throughout this iteration, the following features are intended to be developed: the communication between the mobile devices and the server, encoding of the stream, basic UI for the backend and the mobile application and security measures such as creating emergency session for a given phone number and accepting only video streams from such phone number.

5.2.1.2 Starting a video stream Once a phone call with the emergency center has begun, the user has the option to start the application and choose to start streaming video. The operator will be able to see what the callers phone is capturing live on his screen. Apart from the user initiating the call, the operator will also be able to request the start of a video stream by sending a notification to the users phone.

Before a video stream can be initiated, the operator has to enter the callers phone number into the system. This will match the phone number with the user token into the database and will create a new session with a 2 minute time limit. The video stream has to be started during the time limit, or the session will expire and the user will not be able to start streaming. In such case, the operator has to initiate a new session for the user. This is done in order to check whether the user has the application installed on his phone and will minimize fake calls or misclicks by users.

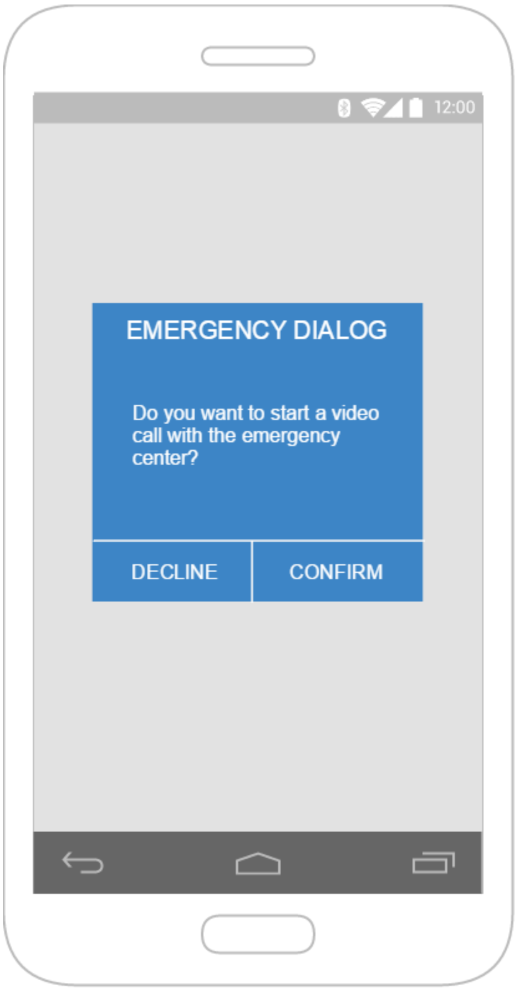


Sequence diagram 1. The operator creates a session.

There are two options in order to start the stream: Either the user opens the app and starts the stream manually, or the operator requests the sending of a notification to the users phone. After the users phone receives the notification, it will display a confirm dialog, which will start the video stream on confirmation.

In the second case, the server sends the notification through the Google Cloud Messaging (GCM) service. In order to do this, the server matches the phone number of the caller with its GCM notification_id in the database and creates a message with a structure as shown below. The message is then sent and handled by the GCM service.

```
...
"data":
      {
          "message_type":"video_call",
          "session_created_timestamp":"DATETIME",
          "session_time_limit":"TIME"
      }
...
```



Once it is received by the mobile phone, the phone checks whether the notification is recent and correct by checking whether session_created_timestamp plus session_time_limit is past the current time. This will handle events, when the GCM service is unable to deliver the message instantly and the message arrives some time after the emergency call. In such case, the confirmation dialog wont be displayed and the received notification will just be ignored.

When received on time, the notification will cause the opening of a confirmation dialog that will prompt the user to accept the video stream and if accepted it fires up the application automatically and starts streaming video, as if the user has started the application by himself. Apart from that, on accept or decline, the mobile phone will notify the server for its choice, in order to update the status of the operators request.

5.2.1.3 Opening the Socket

In order to initiate the video stream a Socket has to be opened, connecting the application with the server. Before the Socket can be opened, the users application needs the token, obtained when the user first installs the application and registers for the service. It is then encoded in the Secure Session Initiation Protocols URI Scheme as shown below. The server then checks whether the token matches any opened video stream session and will either accept it and start the stream, or will decline it.

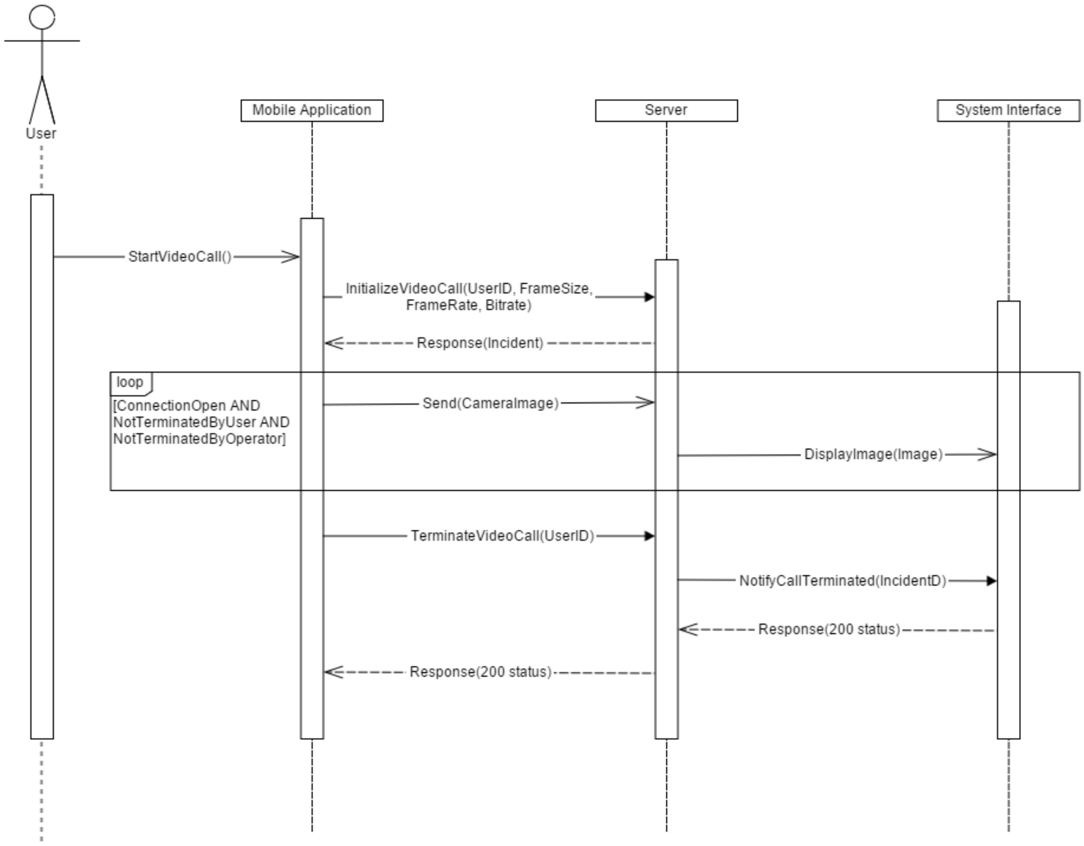
```
sips:<token>@<server_url>:<server_port>
```

If accepted, a Real-time Transport Protocol (RTP) [40] socket is opened between the client and the server. It is designed for handling audio and video communication over the network. To provide a reliable transfer of data, RTP provides error correction algorithms, used for lost packages, and integrated flow control, which ensures that packages are received in order. The above protocols provide Quality of Service (QoS) feedback, which can be used to adjust the quality of the video stream, based on the current internet connection. The protocols also provide Payload identification and Frame indication, which can be used to notify the receiver of any encoding, frame rate or frame size changes. The application will stream only video, as the audio call is already established over the phone and the application should not interrupt it.

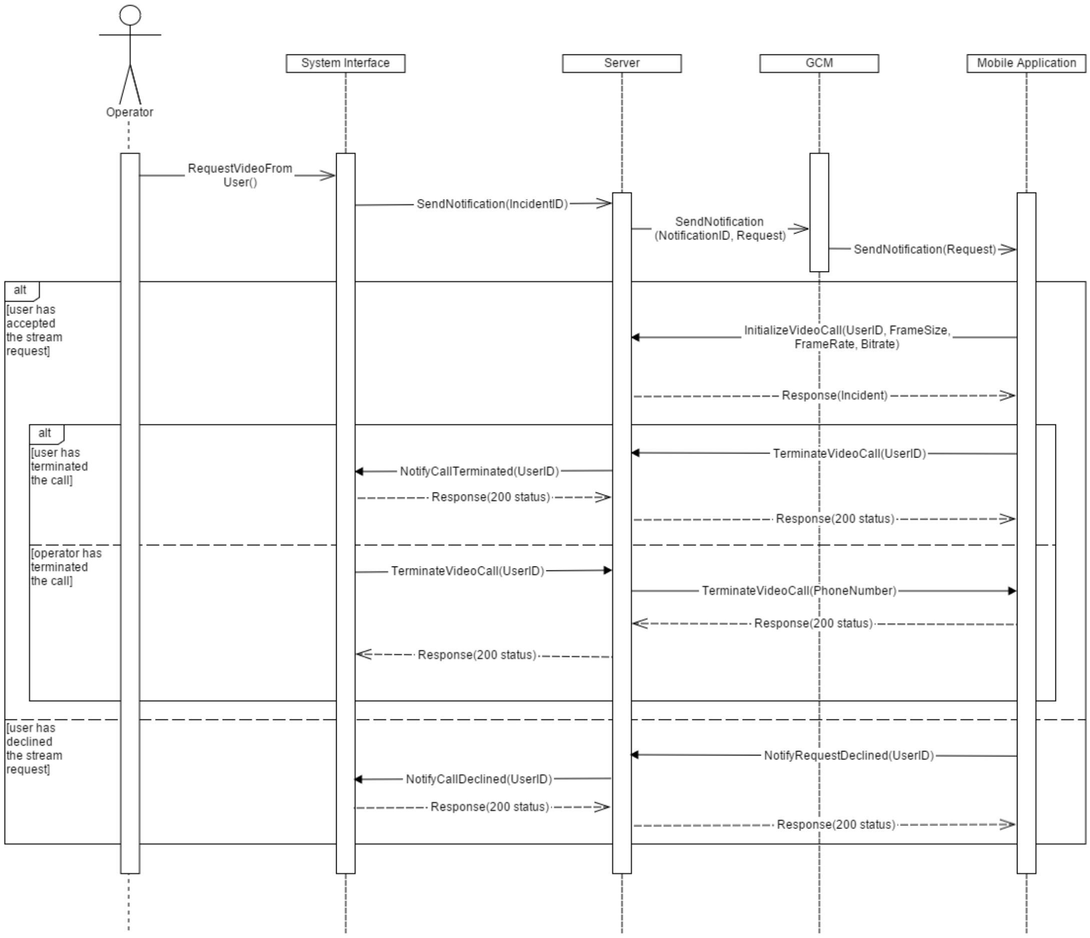
Then the current settings for the stream and camera are sent over: frame size, frame rate and bitrate. Once the socket is opened and configured, the camera is started and streaming begins. The stream is encoded with VP9. We have chosen to use VP9 as a video compression technology over one of the most popular encoding techniques: H. 265, because VP9 is said to be more reliable for streaming [41]. VP9 is open source and royalty free video encoding format being developed by Google [42]. Youtube uses it for 4K resolution content. Matt Frost [43], senior business product manager for the Chrome Web Media Team, addresses audience at Google I/O: People watch more than 4 billion YouTube videos a day and the company streams more than 6 billion hours of video each month. "With a codec as good as VP9, we can significantly increase the size of the Internet," Frost said. "We can significantly increase the speed of the Internet". Based on statistics, VP9 doubles the quality of its predecessor VP8. Google in its own performance comparison claims that VP9 achieves over 50% lower size of the video for the same quality when compared to H.264 [44]. This however, comes with the price of the compressing algorithm requiring more processing power.

On the server side, when the socket is opened and the initial settings are received, the system starts processing the data. Each packet is then decompressed and rendered onto the systems interface.

The user and the operator are able to stop the video stream at any time, using the on-screen navigation on the app or the system interface, without interrupting the call itself.



Sequence diagram 2. The user starts a video stream.



Sequence diagram 3. The operator requests the start of a video call.

5.3 CPR System

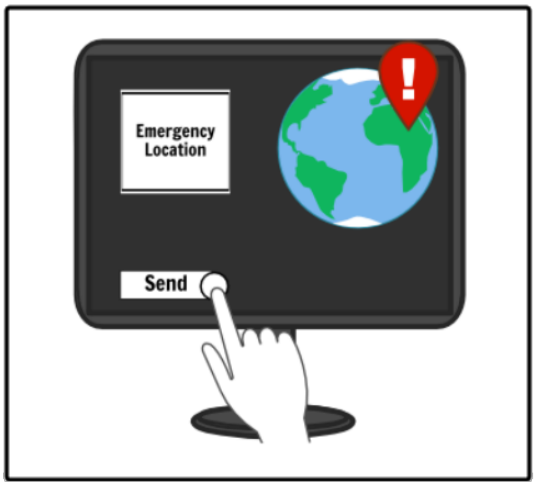
5.3.1 Iteration 1 - Basic Functionality

5.3.1.1 Aims This iteration develops the core functionality of the CPR system; it is intended to be the simplest application possible that has the required functionality. Subsequent iterations will refine and build on top of its services and components.

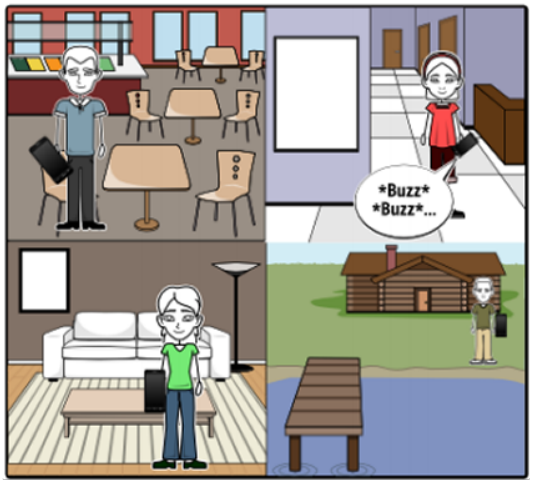
At its most basic level, the system needs to be able to send a message to a responders devices to notify them of the location of an emergency. This individual should be the closest person in the database to the incident. They are then expected to attend the emergency location and provide assistance where necessary. The individual is assumed to have the appropriate qualifications and training which has been verified, and have willingly signed up for the service.

5.3.1.2 Storyboard This is a storyboard to help visualise the intended functionality of the system.

The emergency Dispatcher gets a call about an emergency situation.

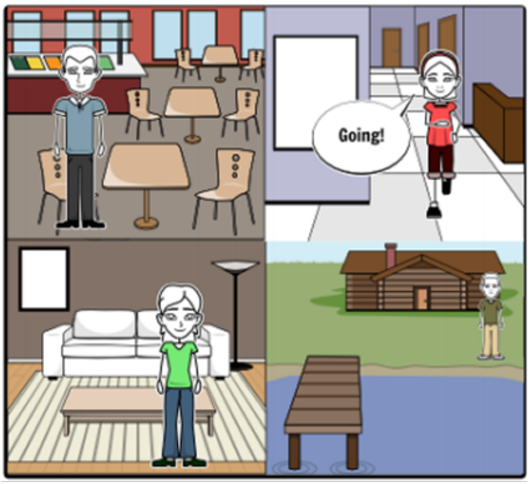


It is an appropriate emergency to ask for help from anyone nearby. The dispatcher sends the help request.

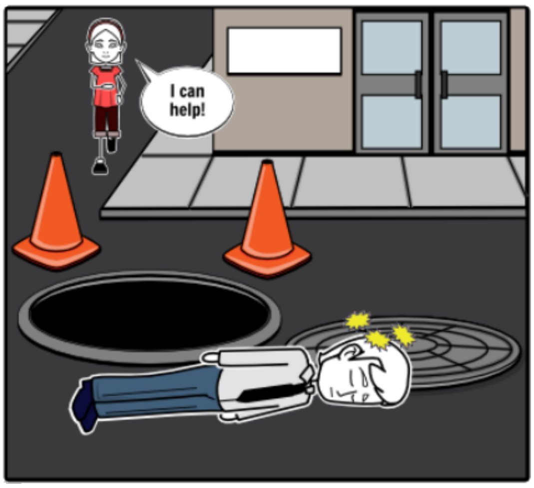


The closest person to the emergency is alerted to the situation.

They are shown the location.



If the responder is available to help they make their way to the emergency.



Once they arrive they help as appropriate.

1. The emergency services receive a call detailing a medical emergency which is triaged.
2. While waiting for a response vehicle to arrive, the dispatcher has the option to send the emergency location to a trained responder (first aid or CPR) who may be able to attend quicker than the emergency services. This option to request this responders assistance should be limited to certain situations and locations so that responders are not sent to potentially dangerous environments.
3. Once the request has been sent, the responder is notified that an emergency requires their presence. The notification shows the responder the location of the incident
4. The responder then decides if they are able to attend the emergency to help the situation and makes their own way there.

In theory, the responder gets to the location quicker than an emergency response and they are able to provide first aid to those who need it until the ambulance arrives.

5.3.1.3 Sending Notifications In order for responders to attend an emergency, they need to know where it is. The standard way of doing this is to send a latitude and longitude, which we will include in the payload section of a GCM message. This information can be directly passed to all mapping applications to plot the location. The structure of the message data should be as follows:

```
...
"data":
      {
          "message_type":"new_emergency",
          "longitude":"VALUE",
          "latitude":"VALUE"
      }
...
```

In this message the `"message_type"` field describes what kind of message this is. To send a new emergency to the user the value of this field should be `"new_emergency"`. This is checked when the message is received and the appropriate action is taken (i.e. a new emergency alert is created). The alternative is `"close_emergency"` as the value and this is used when the dispatcher wishes to cancel the assistance of the user either because they are no longer required or the casualty has been dealt with and the emergency case closed (see App CPR pane for more details).

The other data we need to send is a coordinate pair corresponding to the location of the emergency, (where longitude and latitude values are in the range of [-180, 180] and [-90, 90] respectively [16]) as this is a standard way of referencing a location and is the coordinate system major map applications use. This will allow us to parse the coordinates directly into a map app on the device to show the user where to go. This can be easily done in two ways, one is opening the default maps app with the location entered into a geo-URI [17] and opening it. This URI contains the coordinates and when called opens the default map with the location shown as a drop point (Example: `"geo:0,0?q=34.99,-106.61(Emergency)"`). The second option is to incorporate the Maps API into our app, this would allows us a greater degree of control on what the user can see and also disable features (such as removing the emergency marker). The implementation of this is relatively easy and adding a marker simply uses the `LatLng` data type [18].

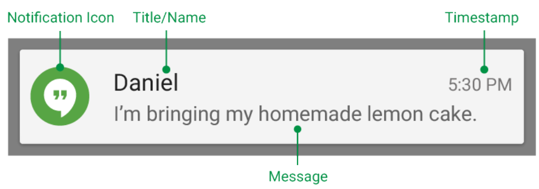
```
public void onMapReady(GoogleMap map) {
    map.addMarker(new MarkerOptions()
        .position(new LatLng(10, 10))
        .title("Emergency"));
}
```

Integrating a map into our app is the route we have decided to go down as this allows us greater granularity and control over what the user sees and is able to do. This should be shown to the user when they click on an emergency notification.

5.3.1.4 Notification Interaction and Alert When an emergency is sent to the phone the user will need to be notified immediately so that they can attend the scene as quickly as possible. The first step towards doing this is to create a popup notification so that the user can immediately see (either on top of other applications or on the lock screen) that their assistance is required.

There are two main factors of information that the user needs to be informed of, the first is that their assistance is required, the presence of this notification does imply this however it should also be stated somewhere in the notification that this is an emergency situation. The second is the location of the emergency, for this we have identified two main points of information that are useful to the user. The first is the address of the emergency location, this should be present so that if the user knows the area around them they can instantly recognise where they are needed and start making their way towards it. It also allows them to give directions, say to a taxi driver, directly from the notification without having to open the app at all. The second is the distance to the emergency that the user is currently This information is very useful to the user as it allows them to instantly gauge how long they will take to attend the scene and make decisions based off this such as the best mode of transport.

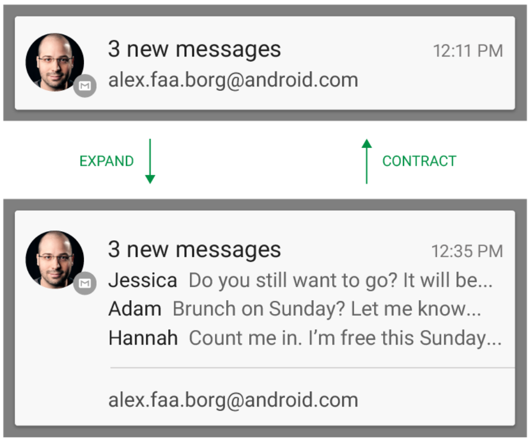
To provide both of these points to the user we will need to use Androids built-in location API, as the information provided to the device is in coordinate form. There are several features which will allow us to obtain the required information, the first is in the Geocoder class of the location API [19]. The method `getFromLocation()` rmation, the first is in the Geocoder class of the location API [19]. The method getFromLocation()takes a coordinate pair (latitude and longitude) and returns a list of addresses which are known to describe the area immediately surrounding the coordinates. Upon calling this we can then extract the address line by line using the associated methods with the `Address` class [20] and insert them into our notification. We can also use the `Location` class of the location API to get the `distanceTo()` a destination [21]. This takes a location as a parameter (which we can set the longitude and latitude of) and returns the distance in meters to the destination from the users current location.



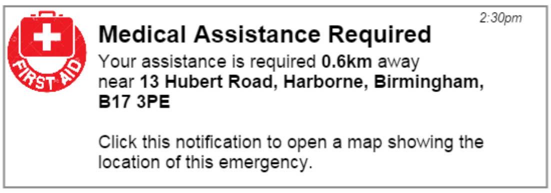
Android supports a variety of notification formats [22] however they all have four common features, a Title/Name, Notification Icon, Timestamp and a Message. Our notification should also have these features, the title should be a common one (for all emergency notifications) which relates to the fact that the users assistance is required at an emergency. A title similar to Medical Assistance Required should both inform the user of what the notification is for and grab their attention.

The notification icon should also be striking to grab their attention, and also needs to be unique to the app and tell the user, at a glance, the nature of the notification. Something similar to this image would be best as the image is striking and so should catch the users attention while being unique among other notification icons and being related to the theme of the notification.

The timestamp of the notification should be set to the time that the push notification was received and the message body should contain both the distance to and the address of the emergency. Android allows expandable layouts [22] which allows the notification to be minimal initially and expand when clicked on. The minimal layout still contains both the title and the icon (along with a line of text) which means the user can still instantly see what the notification is about and can click on it to expand the description (which contains the address).



Minimal notification



Maximized notification

This is a design shown how our notification might look to the user on either the lock screen or while actively using the phone. Upon clicking this notification the app will be launched and show a CPR pane which would not be normally accessible. This pane contains a map with the location of the emergency shown clearly with a marker, the users current location and a directions button which will plot a route to the emergency from the users current location.

5.3.1.5 Cancel Notification The dispatchers should also be able to cancel the assistance request remotely for a variety of reasons such as the emergency has been dealt with or paramedics have already arrived to name a few. To do this we will utilise the collapsible feature of messages sent over GCM [23] which allows us to overwrite previous messages with the same `collapse_key`. We can combine this with the field in the payload of the message `"message_type"` which is set to `"new_emergency"` when the dispatcher wishes to ask for assistance. The new message should have a `"message_type":"close_emergency"` so that, when the message is read the message type should be checked and the appropriate action taken, in this case it should, if its currently accessible, hide the CPR pane and clear any associated data. A message field should also be present which will be displayed in the message field of the notification.

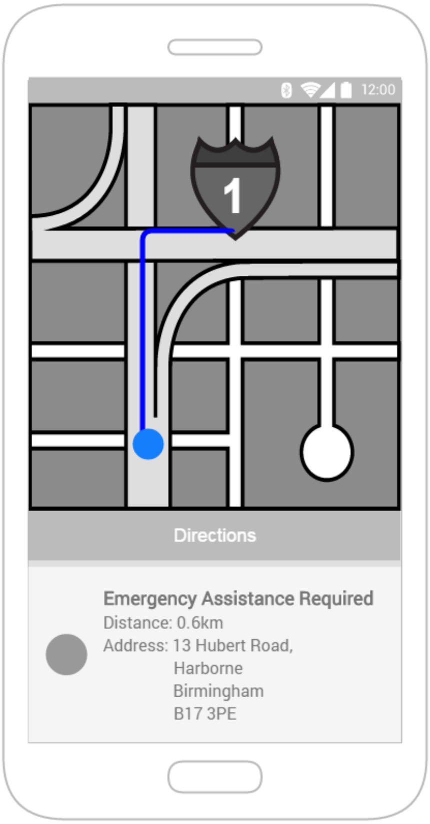
```
...
"data":
     {
         "message_type":"close_emergency",
```

```
        "message":"Emergency resolved"
    }
...
```

5.3.1.6 Notification Sound of the App This was the first thing which we thought the user would like to customize as this has been commonplace among phones for years and our users would expect to be able to do this with our app. However, after some discussion we thought about how this might negatively affect the users ability to be alerted to incoming emergencies. For example, if a user was allowed to set their own notification sounds and they chose one that was the same (or similar) to their message alert or ringtone then they might be more inclined to ignore the phone for a period of time believing it was just a regular notification. This is more likely as our app is likely to be only used occasionally when the individual is the closest responder to an emergency, it will be a case of setup and forget for most users who might even have forgotten they installed it until they are being asked to help.

For this reason we have decided against allowing the user to change the notification sounds. We intend to use a custom alert noise which is unlike any other provided by default on the phone. We believe that keeping this sound is more beneficial than allowing the user more personalisation as they will not recognise the sound and therefore are more likely to check why their phone is making a noise.

5.3.1.7 CPR App Pane This is the design prototype for the CPR pane of the application. By default (i.e if its opened without an emergency) this shows a blank map and information pane which, when an emergency notification is received from the dispatchers, changes to display the information for that emergency. Upon receiving the notification (with the payload data) the coordinates are sent to this app and become the destination marker of the map.



The associated information is displayed at the bottom third of the pane which can be obtained (as in the Notification Interaction and Alert section) from coordinates provided using the location API. The distance measurement should be re-calculated at a set interval so that the value is updated while moving.

The map also contains a Directions button which, when pressed, will start navigation assistance from the users current location to the emergency.

We have decided to implement an in-app map view rather than opening the directions in the default map app because it gives us a greater degree of control over what we can show the user and allow them to interact with. For example it allows us to disable the draggable state of the marker [18] so that the user cannot accidentally move it to the wrong location and then delay their response time. It also allows us to display our own content on the screen (in the lower third) at the same time that the user is getting directions to the emergency. This is currently displaying the address and distance of the emergency from the user however it could be used to display extra information as appropriate.

There are two ways for this information to be hidden once again. The first is that the phone receives a new notification from the dispatchers which contains the same `collapse_key` as this emergency (detailed in Cancel Notification) in this case it should clear any associated data from the app (the destination marker and info pane). The second way is when the app is first loaded a timeout timer should be started which contains a reasonable amount of time to allow the user to attend the scene, but once expired should hide this pane from view and clear any previous emergency data. A timer of 60 minutes should be enough time for any first responder to attend a scene and pass the casualty over to the emergency services (this time should in fact be more than enough). This is so that if, for some reason, the collapse message is not received the information will be removed on its own accord

5.3.1.8 GPS Location We need to maintain a database of registered users locations so that we can target an individual who is the closest person to the emergency and send a notification directly to them via GCM. There are a number of ways which we could implement this, however we decided to (for this iteration) push the location from the device to our servers on a regular time interval. A time period of 1 hour is what we will use initially, this should be enough to maintain a reasonably accurate location of the user (more accurate when they are stationary for a while and less accurate while traveling) while not heavily impacting battery useage or performance of the device with more frequent updates. While not being a very complex or clever system this will be enough for us to get started and will be sufficient for this iteration.

An alternative implementation would have been to push the location of all emergencies to all phones registered on the service. It would then be upto the phone to determine if they were close enough to alert the user to the emergency. We decided against this method for a few reasons, firstly it does not scale well with a large user base. If this was to be rolled out nationally you could easily be sending hundreds of emergencies to thousands of phones on a daily basis. This would also drastically affect battery life of the users device as it would then have to process each emergency received to determine if it was near them, possibly leading to increased user dissatisfaction and uninstallations of the app. Finally, there would be a lot more data being sent and received in this implementation. Rather than sending each clients location once every 1 hour (for example) you would have to send each emergency to every registered phone which would quickly rack up with multiple emergencies per day.

Other ideas we thought about were having the user subscribe to an area that they are likely to be in (possibly one for work and home) then using this location to send messages to people who are likely to be in the area. This is currently how the Swedish project Project SMS-livrddare [6] is contacting its users. We feel however, that this is not accurate enough as there will be situations when they are not around the area (e.g traveling to another city) and using the GPS location of the phone would allow us to contact users with greater geographically accuracy.

To send this data to our server we decided to use GCMs upstream message feature to send the message via Googles CCM (Cloud Connection Service). We chose this over establishing our own connection for a few reasons but primarily for increased efficiency. GCM upstream messages are sent over the same connection used for receiving, which is managed by the operating system and is left in an always open state. It makes sense to utilize this connection rather than use excess resources which would decrease performance (such as battery) [26]. It is also very easily implemented as the API provided does most of the work that would have to be managed if we were to implement our own connection (e.g. checking if the network is available) [27]. It is also not necessary to establish our own connection as the only data we will be sending to the server (in this iteration) is a GPS coordinate pair for the users location, this can be easily converted into string format and sent as a payload GCM message.

The message should be sent according to the specification detailed in the GCM justification section under Sending Upstream Messages. This details that the sent message will have the users registration ID attached which the server can use to associate this sent location with a registered user. The structure of the message data should be as follows:

```
...
"data":
     {
         "message_type":"location_update",
         "longitude":"VALUE",
         "latitude":"VALUE"
     }
...
```

This message, when received by our server, will then be able to parse the message type to see that it is a new location that's being sent and add it to the database appropriately. The longitude and latitude values are the devices current location and can be obtained by the getLastLocaiton() method of the Location API [28]. Android's Network Location Provider determines user location using cell tower and Wi-Fi signals, providing location information in a way that works indoors and outdoors, responds faster, and uses less battery power. This method returns the most recent location currently available, if another app has recently updated the devices location then this will be used otherwise it will be updated before the most recent value is returned.

5.3.1.9 Database Design

The simplicity of this iteration means that it does not require a complex database. We fully intend to have to make sweeping changes to this design in future iterations, however it should form an easily-expandable base on which to build.

The DBMS used for this system will be Microsoft SQL Server 2014, due to its prevalence in professional applications, spatial data types and comprehensive high-availability features, including redundancy and failover, which are critical for a system as important as this.

5.3.1.9.1 Requirements

- *Registration details of responders*
  This need only be a unique identifier and forename and surname for now - it can be trivially expanded later on, so we will not dwell on implementation details.
- *Responders devices*
  Each responder can (and should) register as many devices as they have to their account, in order to increase the chances of them being successfully notified that their assistance is required. In order to do this, we need to store information about responders devices, in particular the push notification key that allows us to tell GCM (or any alternate service, such as APNS) which device to send notifications to.
- *The locations of these devices at any given time*
  To be able to find the closest responder to the incident, we need to track the locations of all responders devices in as close to real-time as possible. We could only keep the latest position, however maintaining the full history (or at least a certain amount of it) is a trivial change, and could be used in the future to improve the responder selection algorithm.
- *Incidents, and the responder chosen to attend that incident*
  The database needs to be able to store the location of incidents so it can send these to responders. It must also keep a history of which responder attended which incidents for logging purposes.

5.3.1.9.2 Data Types

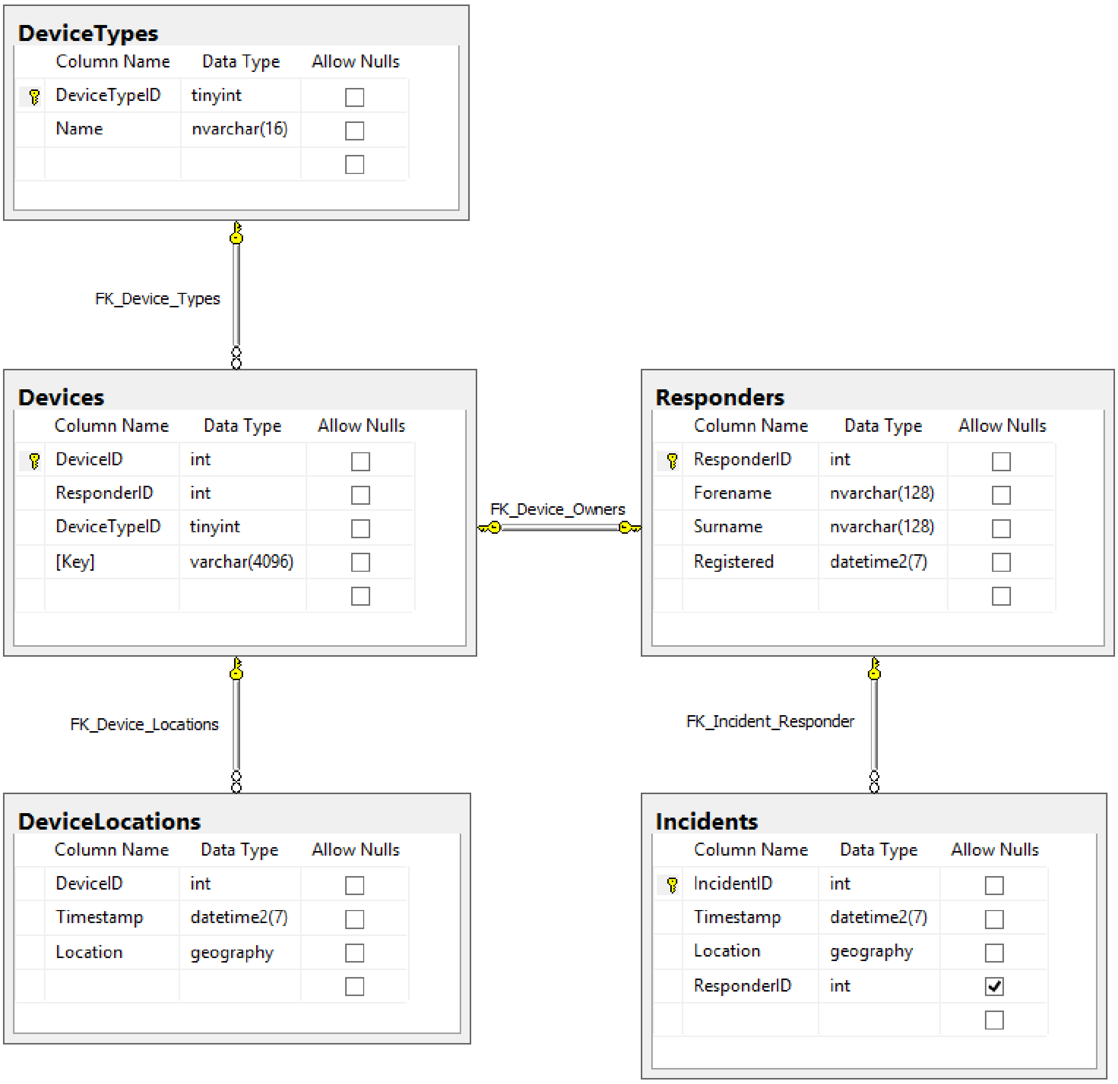
Location data is commonly represented as a latitude and longitude pair, however modern databases have dedicated spatial data types. SQL Server 2014 is no exception; it has the `geography` type for ellipsoidal data. The key `geography` instance type that well be using is `Point`: a 0-dimensional object representing a single location. Well use it to identify the whereabouts of incidents and responders devices. It has the added benefit of supporting elevation, should we need it in the future to increase accuracy of location data.

Primary keys will be `ints`, unless we can guarantee they will not overflow if a smaller width type is used. Unfortunately SQL Server only supports signed types, so 32-bit integers will allow up to 2,147,483,647 (231 - 1) rows per table. Considering there are roughly 65m people in the UK, this is more than enough to store all possible responders. Even at this limit, the type is large enough to handle each of them having 33 registered devices each - also an infeasible number. As already mentioned, there are an estimated 60,000 [2] incidents in the UK every year that this system could be relevant to. Assuming this figure doesnt exceed, an `int` primary key will sustain the system for 35 millenia. The only exception is device types: there are less than 255 services offering push notification facilities, so a `tinyint` will suffice for this purpose.

Device keys are allowed to be up to 4096 bytes, as GCM `registration_ids` can be up to 4KB in size, however in practice they will be much smaller than this - it is this long to handle edge cases.

5.3.1.9.3 Entity Relationship Diagram

Entity relationship diagram of the database.



N.B. `Incidents.ResponderID` is nullable as it is conceivable that an incident might be created, and it take a few moments for a responder to be assigned to it.

5.3.1.10 Selecting the Responder to notify

In this iteration, we are simply choosing the single closest responder to the incident. In terms of the above schema, we need to alert the owner of the device whose latest position is closest to the incident.

We have a table of device locations; the first step is to extract the latest location of each device:

```
SELECT LatestLocations.DeviceID, LatestLocations.Location
FROM (SELECT DeviceLocations.DeviceID, DeviceLocations.Location,
            ROW_NUMBER() OVER (PARTITION BY Devices.DeviceID
                                ORDER BY DeviceLocations.Timestamp DESC) AS Rank
      FROM Devices
         INNER JOIN DeviceLocations
            ON DeviceLocations.DeviceID = Devices.DeviceID
    ) AS LatestLocations
WHERE LatestLocations.Rank = 1
```

As well as spatial data types, SQL Server also includes operations on these types, one of which is `STDistance`, which returns the shortest distance between two `Points`. We can now intersect this with the known location of the incident, producing a list of `DeviceIDs` in ascending order of distance to the incident.

```
-- will hold the location of the incident
DECLARE @incident geography;

-- look up the location of the incident with identity 'x'
SELECT @incident = Location FROM Incidents WHERE IncidentID = x;

-- sort devices by their proximity to the location of the incident
SELECT LatestLocations.DeviceID,
       LatestLocations.Location,
       @incident.STDistance(LatestLocations.Location) AS Proximity
FROM (SELECT DeviceLocations.DeviceID, DeviceLocations.Location,
            ROW_NUMBER() OVER (PARTITION BY Devices.DeviceID
                   ORDER BY DeviceLocations.Timestamp DESC) AS Rank
      FROM Devices
         INNER JOIN DeviceLocations
            ON DeviceLocations.DeviceID = Devices.DeviceID
    ) AS LatestLocations
WHERE LatestLocations.Rank = 1
ORDER BY Proximity ASC;
```

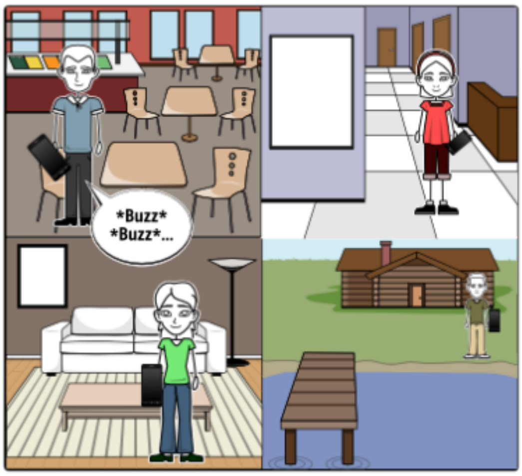
From here, it is trivial to look up the owner of the closest device and notify all of their devices of the emergency.

5.3.2 Iteration 2 - Accept & Decline an emergency

5.3.2.1 Aim This iteration addresses the issue of an individual either not being able to attend an emergency or not responding to the emergency request in a reasonable time. It will also address whether the person contacted is unreachable and how the system should react in that situation.

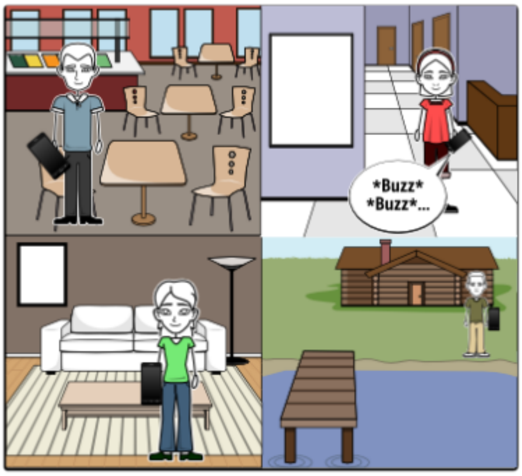
On completion, this iteration should give responders the choice to either accept or decline a request for attendance at an incident, and depending on their decision, the system should be able to react appropriately. It should also address the issue of the contacted user being unreachable after the system sends a help request notification.

5.3.2.2 Storyboard This storyboard visualises the intended functionality of the system.



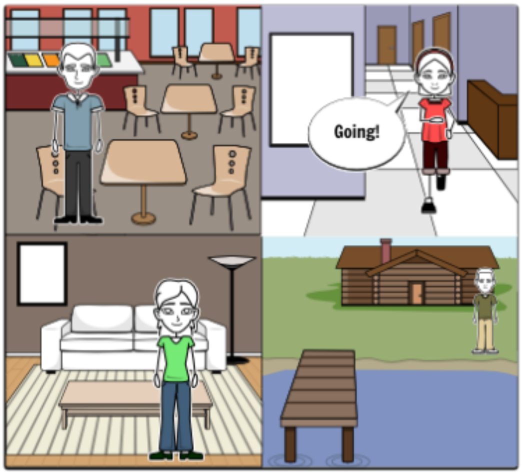
The closest person to the emergency is alerted to the situation.

The user is unable to attend and so declines the request.

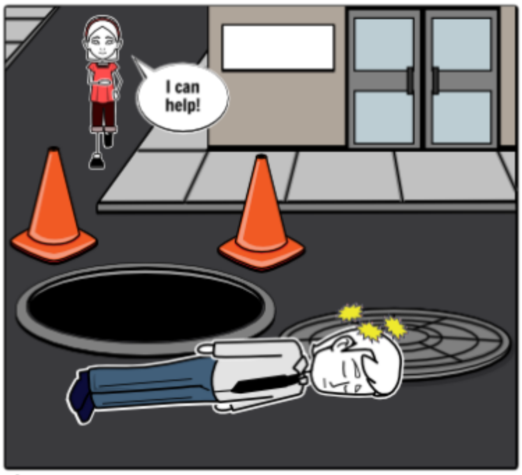


The system finds the next closest person to the emergency and notifies them.

This user is able to attend and responds saying they will accept the request.



The responder makes their way to the emergency.

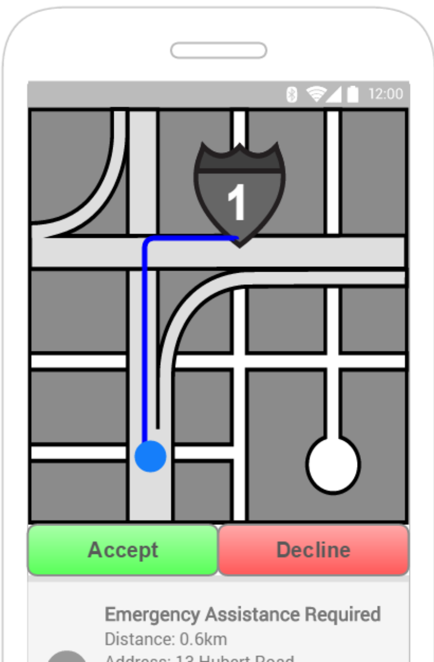


Once they arrive they help as appropriate.

1. The closest person to the emergency is contacted (as per the previous iteration).
2. They are now given the opportunity to accept or decline the request.
3. If they accept the request, they go off to help at the emergency. If they decline, the system contacts the next closest person to the emergency. Subsequent responders are also given the option to accept or decline.
4. This continues until a user says they are able to help or the request is cancelled/resolved by the dispatcher.

This will make our system more robust as we are no longer assuming that the first person contacted can attend the incident. It also gives responders more control over when they can help rather than simply being contacted and their attendance assumed (as in the previous iteration). This reflects a much more realistic use case, as in reality responders will be busy, or their devices will be unavailable, or notifications will be delayed.

5.3.2.3 Confirming Attendance The best way for us to allow the user to tell the server if they intend to respond to an emergency help request is through Accept and Decline buttons. This presents the choice to responders in an obvious way that requires no training or learning time.



These buttons should be present in the app rather than the notification as then responders are able to see more information about the emergency (such as its the location on a map) before making their decision. It is also less likely to be miss-tapped, as the user first has to acknowledge the notification and open the app to be able to respond.

This prototype demonstrates the button placement of the Accept and Decline buttons. They have taken the place of the directions button (which started route guidance to the emergency), as this is not applicable until the user has indicated they are attending the incident. In addition, this placement does not interfere with any other visual elements; all information is still readable and clear.

The Accept and Decline button colours should be red and green respectively to fit in with the standard format for Yes/No buttons, and to encourage responders to accept requests for help. The placement of the Accept button should be to the left, as this makes it harder to mis-tap - the majority of the population are right handed, [29] and this positioning requires more of a stretch to reach.

We have also decided to implement a background timeout for the user to acknowledge a notification. This should start when an emergency request is sent from our server (via GCM) and the notification is displayed. The user will then have 30 seconds to open the notification before the timeout kicks in. If the user does not respond within this time, the notification is automatically declined and the server will send the notification to the next closest person as if the user had declined it themselves.

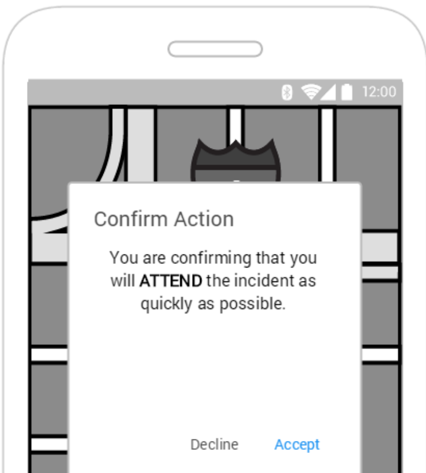
This is implemented because of the time sensitive nature of these type of emergencies. The idea of the first responder is to be able to attend the scene before the emergency services can get there (the target response time being 8 minutes[30]), so any delay in alerting a potential first responder reduces their usefulness. As a result, if the first person contacted does not reply quickly enough then our system will discard them and move onto the next person who might be able to help faster.

This cycles should continue until one of the following conditions has been met:

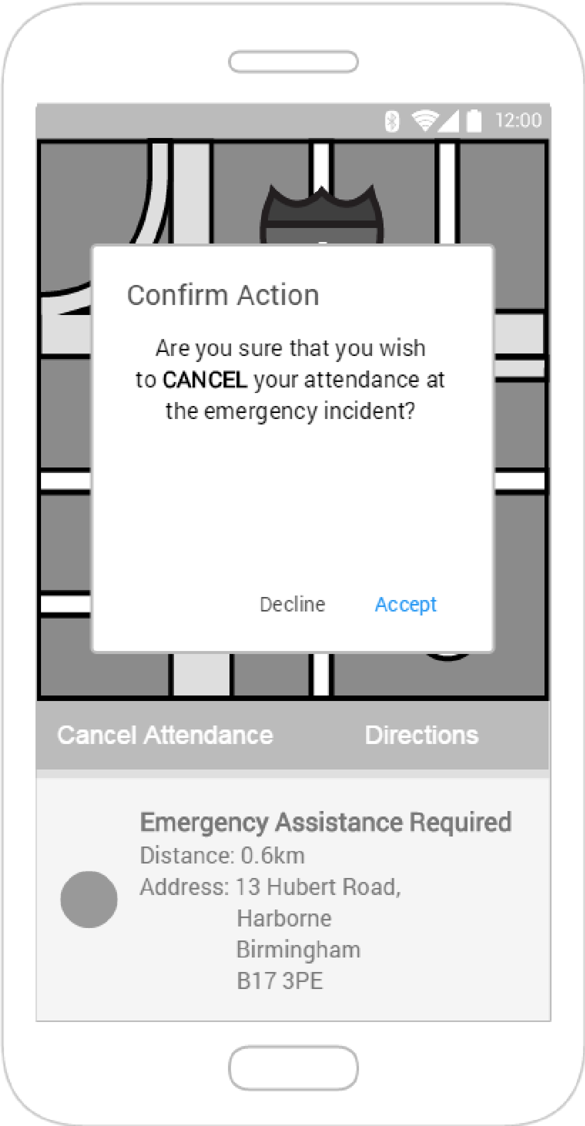
- The 8 minute average response time has passed, after which we expect an emergency response to be either at the incident or very close. After this time the likelihood of a first responder being able to help the injured individual has also decreased dramatically, to the point where it is highly unlikely they would be able to do anything.
- The next individual who is due to be contacted with the emergency is beyond the reasonable response distance for a first responder. This stops the chain of help requests reaching anyone who is too far away to be able to help. This distance should be based off the time it would take an individual to travel to the incident. On average an 8 minute journey would take the traveller around 2-3 miles in an urban environment, and up to 5-6 miles in a more rural setting. Therefore our cut-off distance should be towards the upper end to allow people in rural areas the chance to attend the incident. Around 5 miles should be appropriate.

We cannot use the state of the emergency as a cut-off for our system, as emergency responses tend to not report on the status of their emergency until the casualty has been delivered to a hospital. If we were to use this then our system would still be sending first responders to the emergency after the ambulance had left, wasting their time and reducing the likelihood of them continuing to use the system.

5.3.2.4 Confirmation Messages The Accept and Decline buttons should also have a verification message box to ensure the user intended to select that option. This should clearly state the action (accept or decline) and be placed on the screen so that the bottom information pane is still visible.



Finally, upon the user accepting an incident, the app pane should have a Withdraw button still visible for cases where there was a missclick (even after the verification), or a situation arises where the user is unable to attend (such as unforeseen traffic jam). Upon clicking this button the system should resume as if it were a normal decline, and alert the next user if the end criteria have not been met.



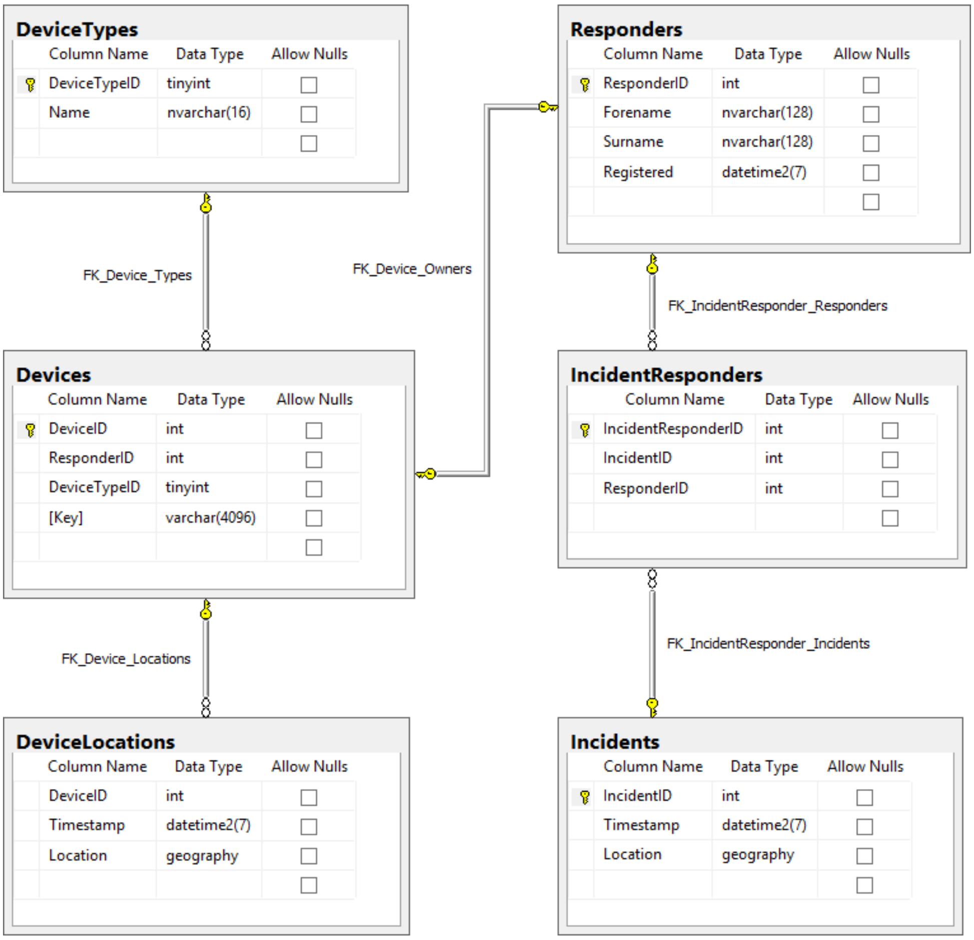
This button should be placed on the left hand side of the screen for the aforementioned reasons, and be next to the Directions button so that it does not interfere with any of the information being displayed to the user.

Upon clicking this button, the user should be presented with a confirmation screen which clearly states what the action they are performing means.

5.3.2.5 Keep playing alert sound until user accepts/declines This is another feature of the app which we discussed and decided on what to implement. Should the phone play a single notification sound or for the sound to keep playing (like an incoming call) until the user accepts or declines it. We decided to play a continuous alert until the user begins to actively use the phone, this means that the alert will play until the phone is turned on (from sleep state), the volume buttons are pressed (standard way to mute an alert on most devices) or the user accepts or declines the emergency alert.

This can be easily implemented as we are able to run any code which we like upon receiving a GCM message for this app. This will allow us to start the alert tone and add hooks into the various stop events which will cancel the playing of the alert.

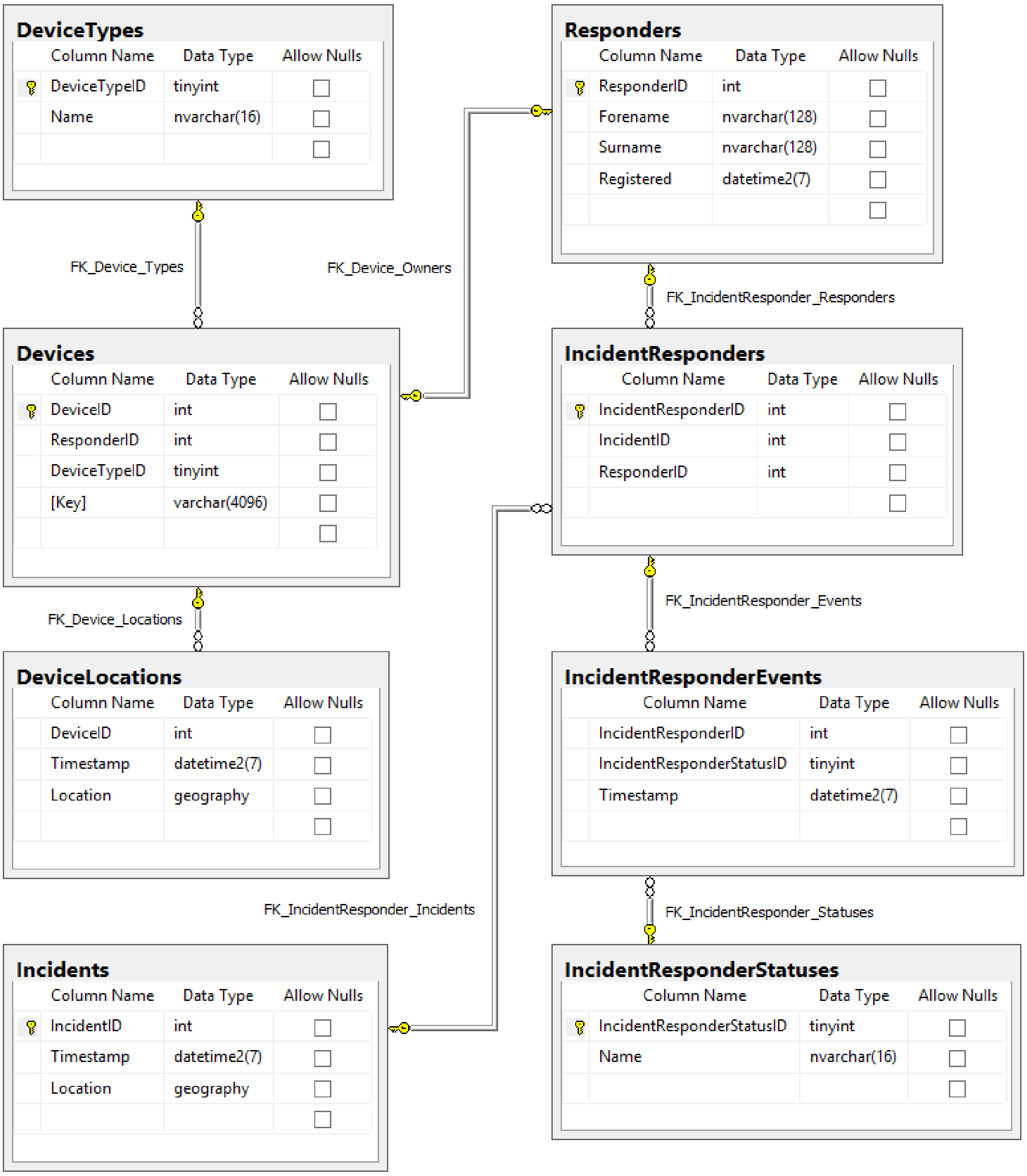
5.3.2.6 Database Modifications This iteration requires the ability to link multiple responders to an incident, which necessitates several changes to the database design. The first step is to remove the `ResponderID` field from the `Incidents` table, as the relationship is now too complex to be implemented in this way. Instead, another table of `IncidentResponders` is needed, which had a foreign key into `Incidents:`



This allows us to keep track of multiple responders attending a single incident, each having an `IncidentResponderID`, which uniquely identifies a responder attending an incident.

Next, we need to be able to keep track of the status of each responder - effectively the progress of their notification, from sent, to accepted, declined or withdrawn, including no response at all.

This information requires another table, which shall be called `IncidentResponderEvents`:



In the schema above, when a responder is selected to attend an incident, a row is inserted into `IncidentResponders`, and notifications for their devices queued for sending. When this completes, a row is inserted into `IncidentResponderEvents` with status sent. From this point on, the aforementioned timeout for the responder to indicate whether they will be attending the incident starts. This timeout should be implemented in the application layer.

When the responder selects accept or decline, the application will insert a new row into `IncidentResponderEvents` with the corresponding status, and similarly for if a responder withdraws their acceptance. If that status was declined or withdrawn, the application must first check the timeout for responder help has not expired, before repeating responder selection, which takes into account declined responders for an incident, making sure they are not selected again. More about this process is detailed in the next section.

5.3.2.7 Selection Algorithm Modifications

This iteration requires selection of the next best responder, which the previous algorithm could not cope with. However, the change is not a difficult one. Given the above database modifications, it is easy to take the set of responders who have declined or withdrawn from a given incident, and subtract this from the set of all potential responders who could attend, removing their devices from the selection step.

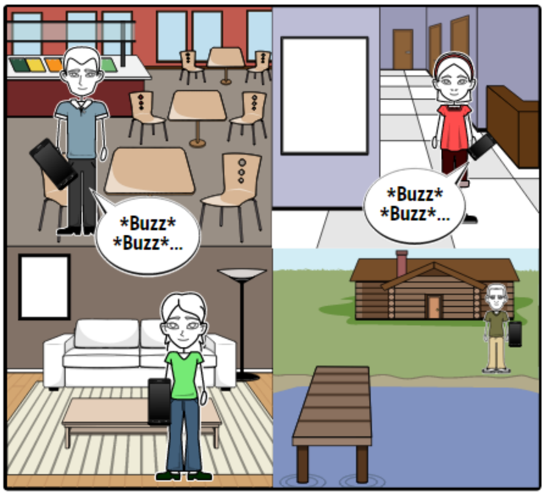
Similarly, introducing a maximum distance, above which devices (and therefore responders) are ignored is straightforward. SQL Servers `STDistance` function returns a distance in metres, so a conversion into miles can be done by dividing the returned value by 1609.344. Armed with this information, its simply a case of adding another `WHERE` condition to the previous device selection query.

5.3.3 Iteration 3 - Improving Responder Alert

5.3.3.1 Aim This iteration addresses the issue of our system currently only contacting one person at a time when searching for an emergency responder. With the timeout of the accept response by the contacted user being (at minimum) 30 seconds, the time between the first person contacted and eventually finding an available responder could be many minutes (which would eat into the 8 minute ambulance response time [30]).

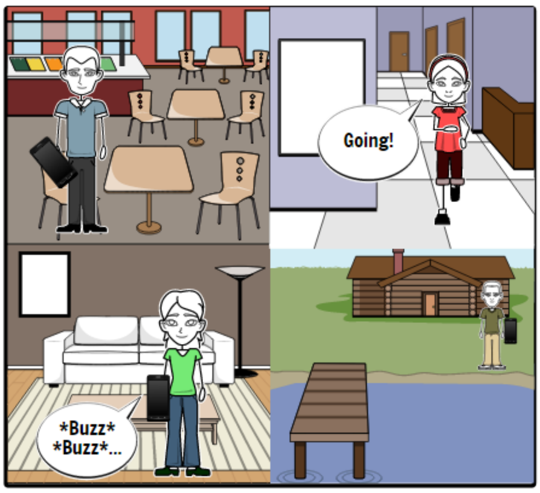
This iteration aims to reduce this concern by adding a parallel search (in the same vein as the previous iteration) so that multiple responders can be contacted simultaneously increasing the likelihood of contacting someone who is available and reducing the responder response time.

5.3.3.2 Storyboard This storyboard visualises the intended functionality of the system.

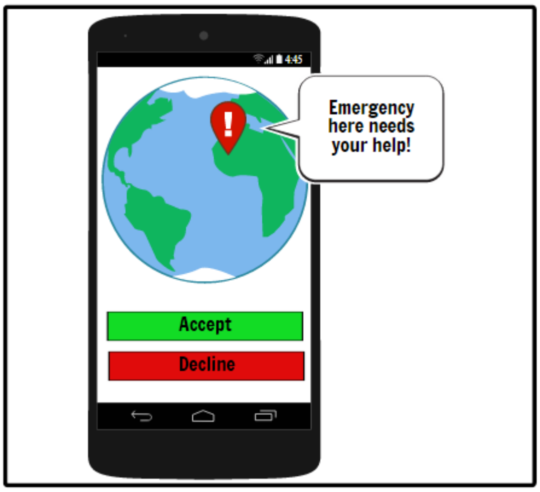


Multiple responders are simultaneously alerted to the incident.

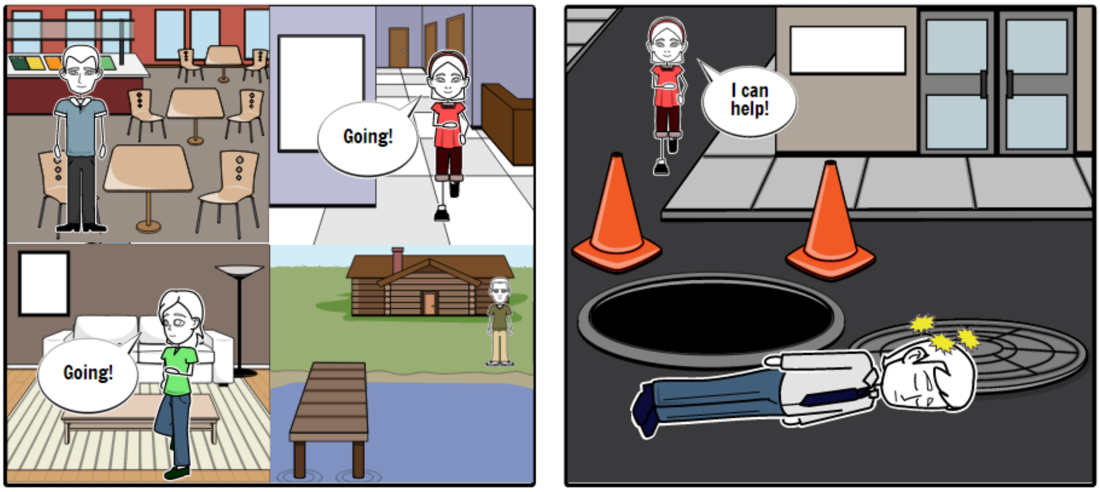
When a responder accepts that search thread is stopped.



The parallel search moves onto the next person while the first accepted responder attends the incident.



The parallel search's continue until the end conditions are met.



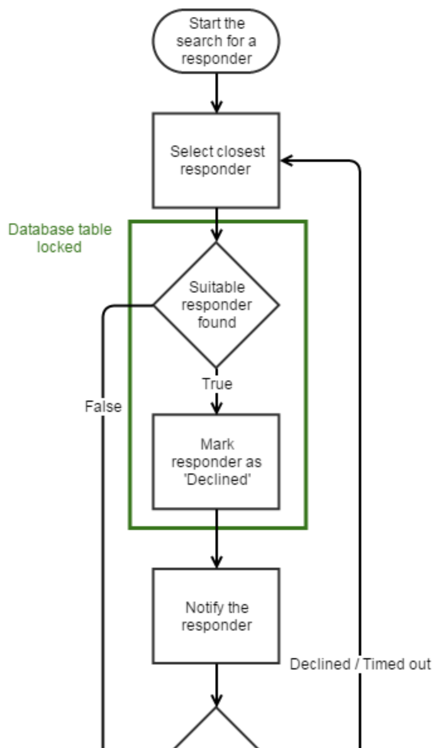
One or more responders are able to attend the incident — Once they arrive they help as appropriate.

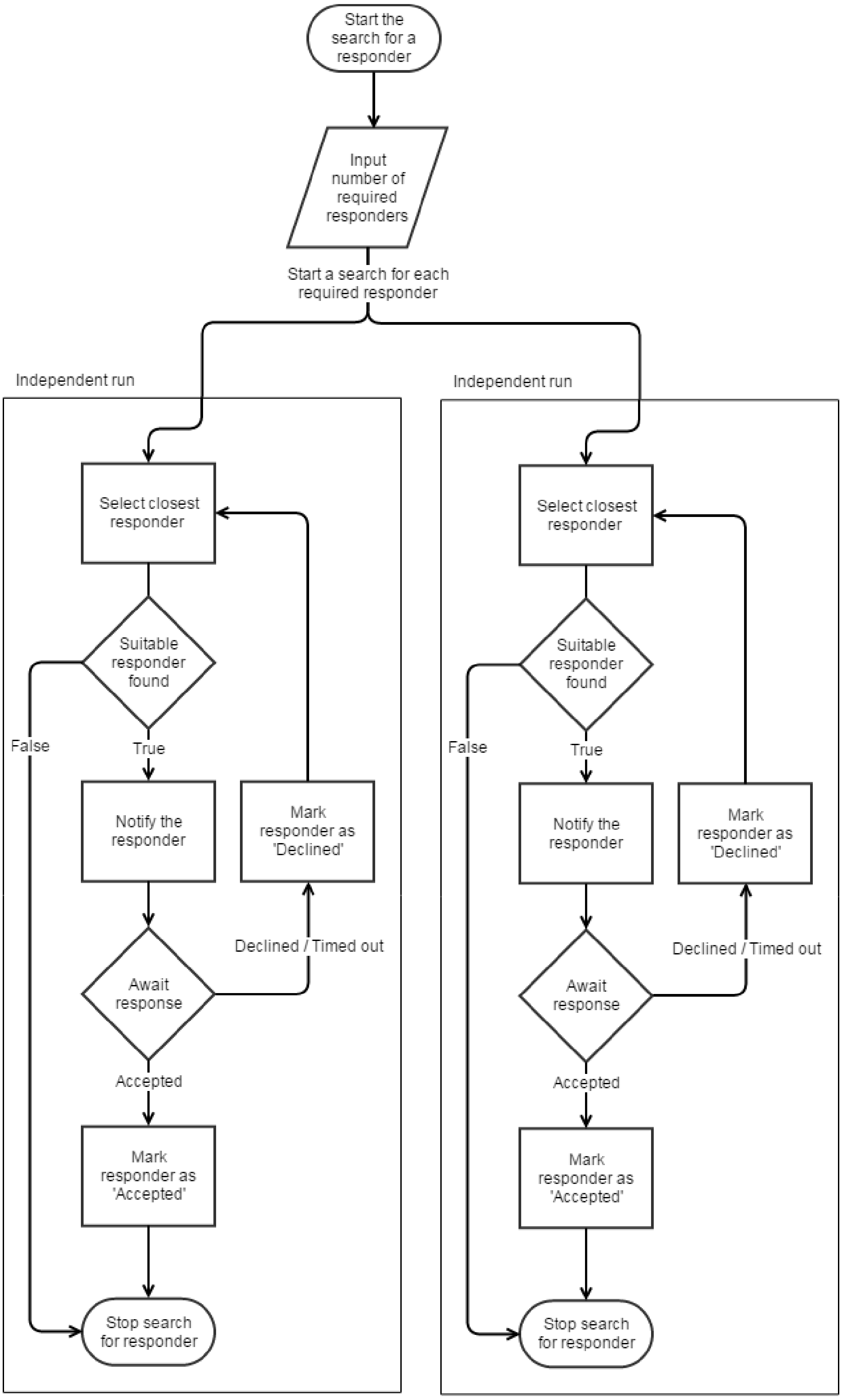
1. Multiple parallel searches for responders are started. Each stream selects from the same group of responders (without overlap).
2. The responder is given the opportunity to accept or decline the request (as in the previous iteration).
3. If they accept the request, they go off to help at the emergency. If they decline, the system contacts the next closest person to the emergency. Subsequent responders are also given the option to accept or decline.
4. This continues until a user says they are able to help or the request is cancelled/resolved by the dispatcher.

This will help our system to more quickly find an available responder to attend an incident as we are benefiting from the effects of parallelization. It also allows us to send multiple responders to an incident so that if one was delayed on route there is already another responder who is actively attending. It would also allow the dispatcher to specify that they required many responders, for example a large scale emergency with many casualties.

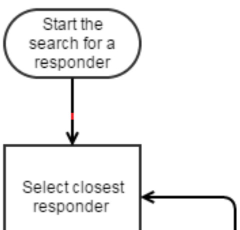
5.3.3.3 Parallel Searches In order to implement the ability to perform parallel searches for responders we need to first look at the way in which the process for searching and selecting a responder is done. Below is a flowchart showing the basic layout of searching for a responder on the server side. We can see how the workflow lends itself to parallelization as there is a single flow from top to bottom which, once started, has no outside influences and will terminate by itself once a responder has been found or there are no more suitable responders to ask (based off the criteria in the previous iteration).

This means that with some refactoring we can set up multiple runs of this search alongside each other with little effort.





The above diagram shows how, after the number of required responders is specified, the searches can be run independently of each other and terminate on their own. Implementing this would require extracting the initial setup of the search (passing the details such as the location) and calling a overseeing method which then start the required number of parallel searches. As each search is using a common database for selecting and marking users decisions (using the methods described in the previous iteration) this should eliminate the possibility of a parallel search selecting a responder for an event who has already been contacted.

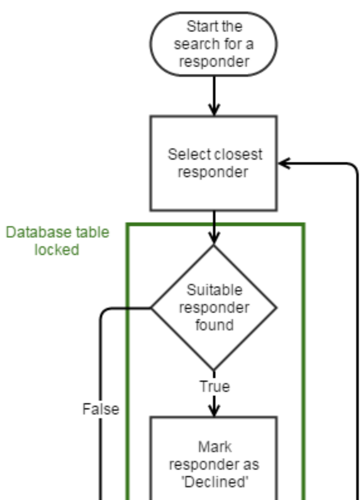


We do need to take measures to prevent some problems. The first being that we need to ensure that when parallel searches come to select a new responder to contact they do not select the same responder in the time between one search selecting the user and marking them as available or unavailable. The red box on the below diagram illustrates the current area of overlap between a responder being selected by a search and that responder being marked by that search (i.e. the area where two searches could select the same responder to contact).

To counter this we need to implement two things, first we need to mark a chosen responder as soon as they have been selected. Currently a responder is only marked as chosen after they have accepted or declined (minimum of 30 seconds) so if we were to leave it like this any parallel searches would have to wait on this mark to continue.

This can be implemented right after the selection has been made and before the notification is sent to the user minimizing the time that a selected user is not marked. We could add another table in the database for SelectedResponders however we already have a system in place to differentiate between selected and non-selected responders (the accepted/declined mark). We could simply mark a responder as declined as soon as we have selected them as this does not impact the flow from that point on (contacting the user and obtaining a response) and then change this mark to Accepted if they accept. If they decline or the operation time out then we have already marked them as Declined and simply return to the start of the search process.

In order to further reduce same responder selection we can close the window for searches to overlap completely by locking the database table while one search is selecting a responder until they have marked them as selected. This would mean that two parallel searches that ran at exactly the same time would not be able to access the database table and so one would have to wait until the other finished and unlocked the table before continuing. Introducing a lock also introduces a form of staggered startup so that at the initial responder selection for each parallel search there would not be an occasion where they would both be able to select the same initial user to start as they could be running very close (time wise) together and have a greater chance of overlapping.



The diagram left shows the revised series of stages for the search to complete and displays the time that the database table would need to be locked for. As you can see it is substantially less than the overlap time from before and both of the operations inside the lock should be completed very swiftly. As soon as the lock is lifted any other searches looking for a responder are free to search and the responder selected by the search that owned the lock has marked that responder so no other searches will select it.

This should fix all of the problems when implementing this feature and allow us to speed up the selection process and allow multiple responders to be contacted.

5.3.3.4 Usage

This feature will be used in two distinct ways, firstly it gives the dispatcher the ability to request more assistance for larger emergencies in the form of more responders being directed to the site. A use case for this could be a car crash with multiple injured people where one responder would be quickly overwhelmed and unable to help more than a few people. This request will be part of the API and can be built into the dispatchers system to increase the number of responders as they see appropriate.

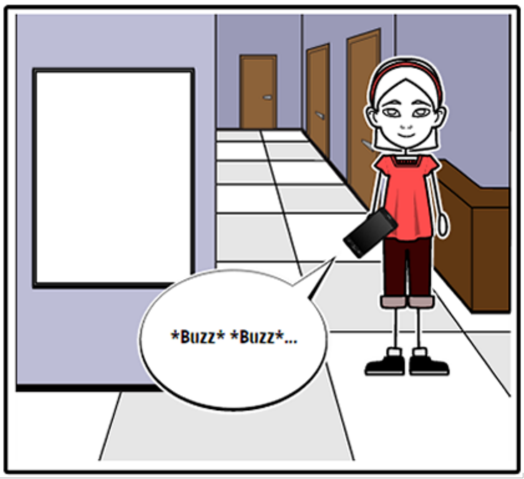
The system should also be used and, by default, contact two people for each emergency. This would mean that more potential responders are contacted faster than in the old system (through the parallelization) and it should end with two responders being sent to the incident. This is useful as it increases the chance that one of them arrives before the paramedics (for example one might be stuck due to unforeseen circumstances) and the risk of the responders not getting there is halved (split between the two).

We feel that it is acceptable to send two responders to an incident by default as it firstly decreases the risk that the responder does not arrive. It also means that, if both responders arrive, they can help each other with the casualty until the paramedics relieve them. We feel that sending two responders to the same incident is acceptable and that they should not mind not being the only one sent as the health of the casualty should take priority.

5.3.4 Iteration 4 - Context of the emergency

5.3.4.1 Aim This iteration addresses the issue of the responders currently going out to an emergency blind to what they may encounter. This iteration will address this by supplying relevant information to the responder when they are alerted to an incident. This information will help them identify the patient when the arrive on site and alert them to any critical information that may be of help.

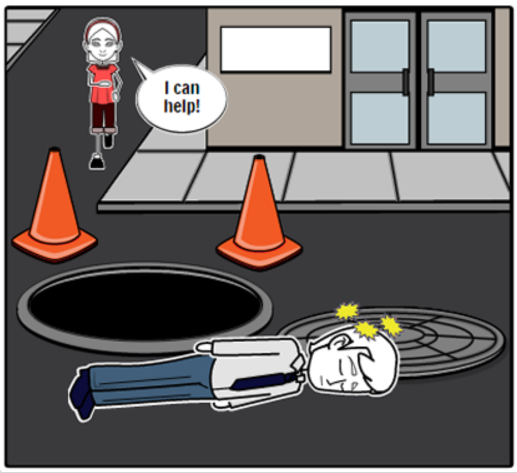
5.3.4.2 Storyboard This storyboard visualises the intended functionality of the system.



The responder is alerted that they are required at an emergency.

Any appropriate information is displayed to the responder through the app.

The responder makes their way to the emergency, able to mentally prepare for what they need to do.



They arrive and are able to locate the casualty easily, then help as appropriate.

1. A responder is contacted about an emergency situation near by.
2. The message contains details about what kind of emergency situation they are being asked to help at as well as relevant details of the casualty (if any are available).
3. If they accept the request, they go off to help at the emergency. If they decline, the system contacts the next closest person to the emergency.

4. After accepting they make their way to the emergency knowing what to expect on the scene and being aware of any special precautions they need to take when dealing with the patient.

This will help the responders to emergency situations mentally prepare before they arrive on scene. Information provided to them will alert them to the type of emergency they are attending, allowing them to think about potential first aid that they may need to give. It may also help them to identify casualties better in certain situations, being given a brief description of the casualty (e.g. Sex, Age) for situations where there are multiple casualties (allowing them to go directly to the more serious one).

5.3.4.3 Information When a phone call is received by the emergency services there is a standard set of questions (for a medical emergency) that will be asked by the call taker. The first set of questions, used to ascertain what help is required, are the location of the emergency, the phone number the caller is using and exactly what has happened regarding the emergency call [31]. Once this information is received the call taker is able to send help. Then the call taker will ask the caller for any additional information such as: the patient's age, gender and medical history. Whether the patient is awake/conscious, breathing and if there is any serious bleeding or chest pain; and Details of the injury and how it happened [31].

Out of this information there are two distinct sections. The standard information gained from the questions required before the emergency services are dispatched and the extra information which may or may not be given for a call (dependant on the circumstances). This means that we can rely on the ability to provide some extra information to the emergency responder, such as location (which is already being sent) and a description of exactly what happened. However, in some cases we are able to send additional information as detailed above to help the first responder.

For some of this information we need to be mindful of the data protection of the individual in concern and any policies that may be in place by the organizations implementing our system. The London Ambulance Services policy on disclosure to third parties is We will not disclose your information to third parties (for instance outside the NHS) without your permission unless there are exceptional circumstances [32]. This would mean that without modifications to their data protection policies to either include first responders or class them under the exceptional circumstances, it is unlikely that they would make use of the extra information sending capability of our system. However, sending the description of the incident should be acceptable as this would contain no personal information about the casualty. We have decided to implement the ability to send the additional information in our system for potential future use by the organizations in question after a review of their policies.

A breakdown of the extra information that can be included in a request for a first responder is as follows:
Mandatory information:

- Description of the incident

Voluntary information:

- Patient age
- Patient gender
- Relevant medical history
- Current state of the patient (awake/conscious)
- Breathing
- Serious bleeding and/or chest pain
- Additional details of the injury and how it happened.

The mandatory information should be included in all messages sent to first responders as this information is always given to the call taker of an emergency call. The voluntary information may be given in full or part and as such should be included if applicable (and depending on the data protection policies involved).

5.3.4.4 Notification Details In order to send this new information to the first responders we can utilize the payload feature of GCMs messages again. The modifications would only need to be to the new_emergency messages and there should be a field added for each section of additional information that is to be sent. The structure of the message data should be as follows:

```
...
"data":
     {
         "message_type":"new_emergency",
         "longitude":"VALUE",
         "latitude":"VALUE",
         "incidentDescription":"VALUE",
       //optional after here
         "age":"VALUE",
         "gender":"VALUE",
         "patientState":"VALUE",
         "breathing":"VALUE",
         "bleeding":"VALUE",
         "chestPain":"VALUE",
         "medicalHistory":"VALUE",
         "additionalDetails":"VALUE"
     }
...
```

The incidentDescription field should be present in all new_emergency messages from now on and contain a text description of the incident taken from the call takers notes while answering the call. Any optional fields should be not included in the message if no message data is present for them and then parsed in the app to determine which fields to display to the user.

The datatype of the optional fields are as follows, though all field values are sent as strings the data in them should be in a standard format to be understood by the app: Some of the data has

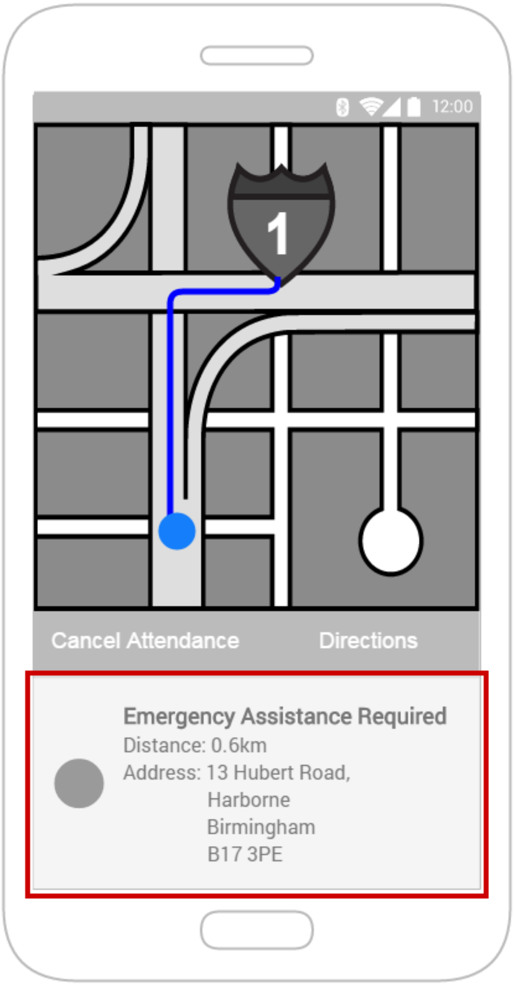
Field Name	Data Type	Example Data
age	Integer (1-3 characters)	"77"
gender	Single Character	"M" or "F"
patientState	Single Character	"C" or "U" (conscious& unconscious)
breathing	Single Character	"T" or "F" (true or false)
bleeding	Single Character	"T" or "F" (true or false)
chestPain	Single Character	"T" or "F" (true or false)
medicalHistory	String	"Asmatic, previous heart attack"
additionalDetails	String	"Casualty fell down flight of stairs"

been compressed into single characters to be sent over the message, this is to help maximize the available space of the GCM payload (which is limited to 4kb) for other fields which may contain long strings. Fields with a single character will have a limited pool of available values which can be parsed on the app to display a more meaningful message to the user.

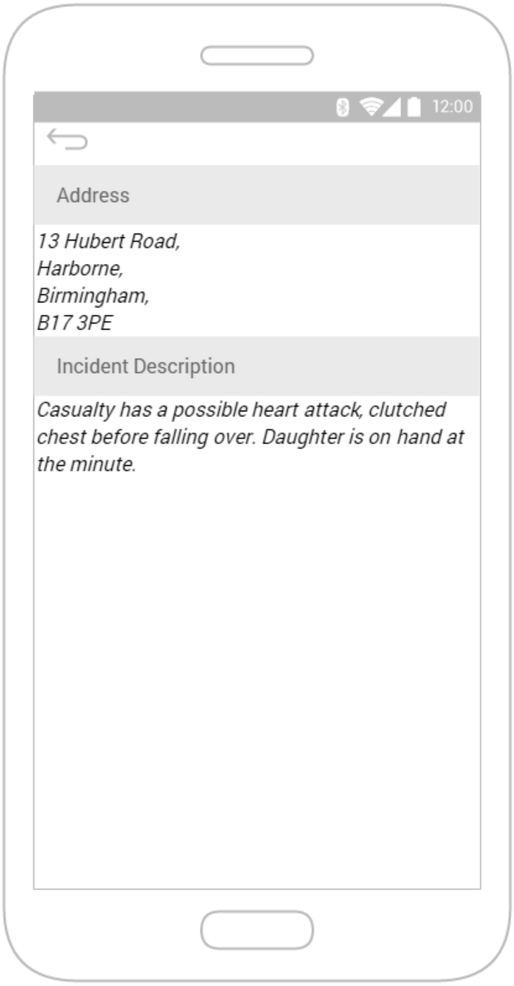
5.3.4.5 App CPR Pane This is the prototype for the additional details display pane. This can be accessed from the home page of the app by pressing on the location details covering the lower third of the pane. We decided to use this as the doorway to accessing the additional information rather than a button because we are already dealing with limited space on the menu bar below the map. We wanted it to be accessible while the accept/decline buttons are visible (which cover the menu bar) and we feel that it is an intuitive way to obtain more information. Many apps utilize the click to explore methodology and this is what users will be expecting from their previous experiences.

The details pane shown below illustrates how an emergency notification with only the essential information will be shown to a user. We maintained the address at the top so that if the user was making their way to the location they would not have to switch back to the previous pane to see the address. The incident description is also present which gives the user some information on the incident and what they might expect.

This pane also has a prominent back button which the user can use to move to the homepage of the app.



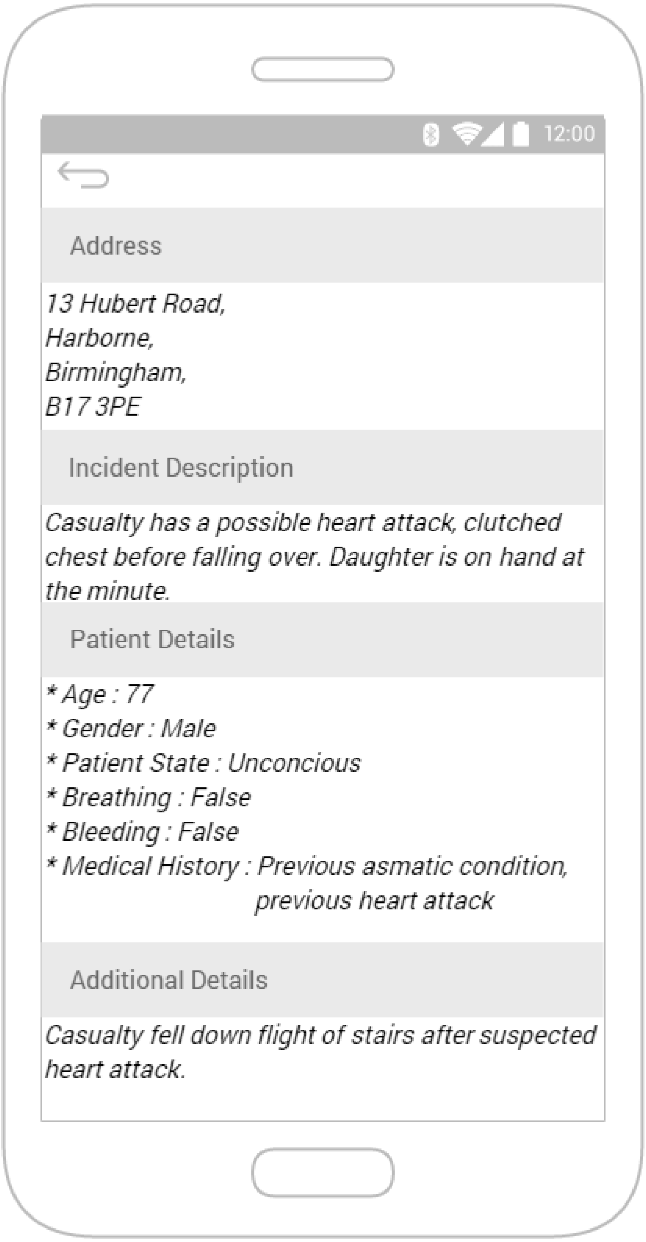
Clicking on the basic 'emergency details' will bring up the in depth details pane.



This shows the most basic level of information provided to the first responder.

A more detailed view is shown in the image below, here much more information has been provided to the user. The information is grouped into common sections and any single characters from the message data have been expanded into their respective words/phrases. Any omitted information is not shown to the user, for example the `chestPain` field is not applicable here as the patient is unconscious and so it was not sent in the message and the app does not display anything regarding this piece of information.

This pane is scrollable so that if a longer description is provided, or in future versions more information is available to be sent, it will all fit on this pane and the user can easily scroll to the information they need.

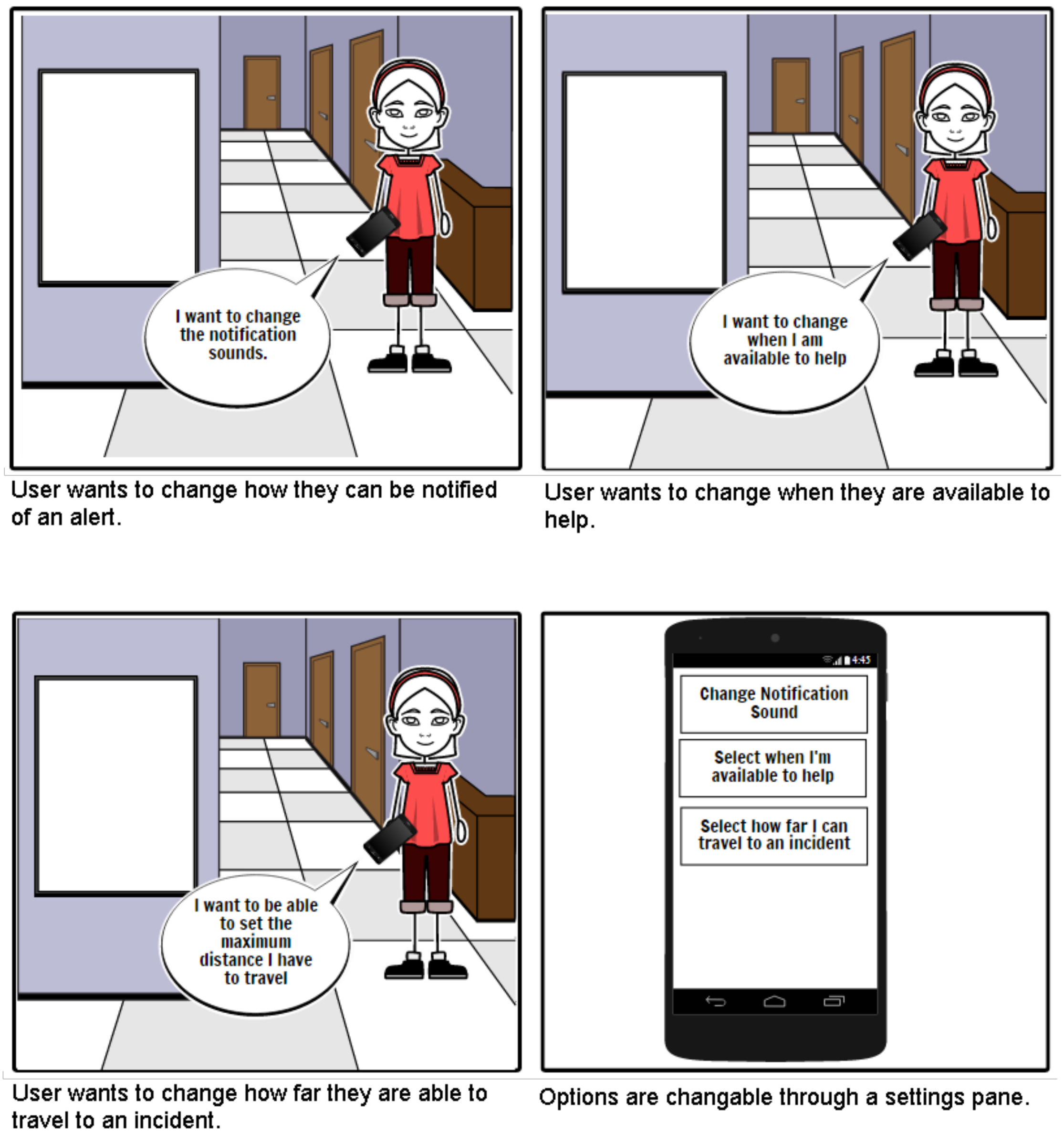


A more in depth details pane showing additional details as provided by the dispatcher.

5.3.5 Iteration 5 - User Preferences

5.3.5.1 Aim This iteration gives the user more flexibility and control over various features that the CPR system offers. The user should be able to change preferences such as when they can be contacted or how they are alerted allowing them to personalise the service to fit into their lifestyle.

5.3.5.2 Storyboard This storyboard visualises the intended functionality of the system.



1. A responder wishes to personalise the app to better suit their lifestyle.
2. The responder can find a variety of options inside a settings menu of the app.

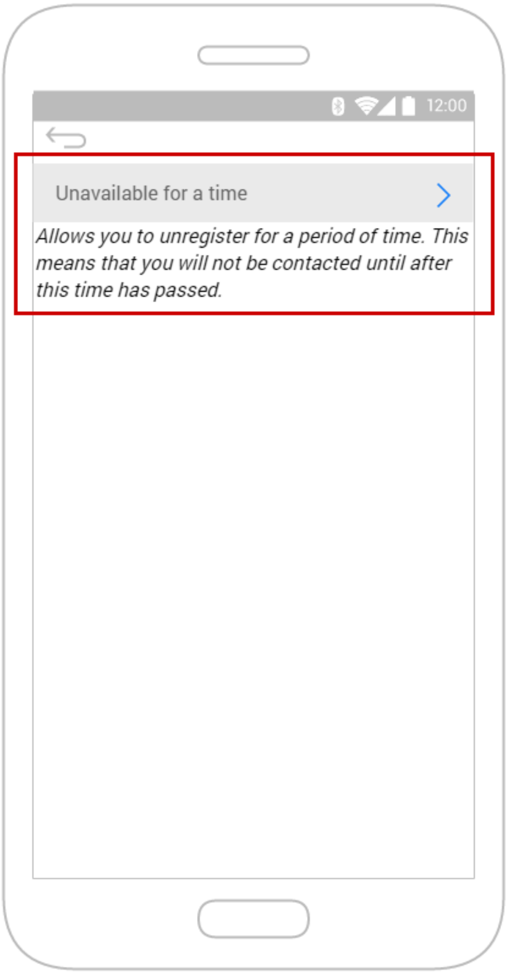
This will help the responders to emergency to personalise how they are contacted to fit into their lifestyle. There should be options to change notification alerts, change how often they can be contacted, set hours they can be contacted along with more. This means that the user has control over when it's convenient for them to be contacted which makes it more likely they will keep using the service improving its efficiency and user base.

The settings menu of the app will be accessible from the default OS settings menu. Here there will be an app-specific pane which will grant access to the settings menu for our app. This means that we do not have to implement an icon on the main app page which would take up space from more important features. Also, the user is unlikely to be regularly changing the settings for this app so they are not needed to accessible from the app itself.

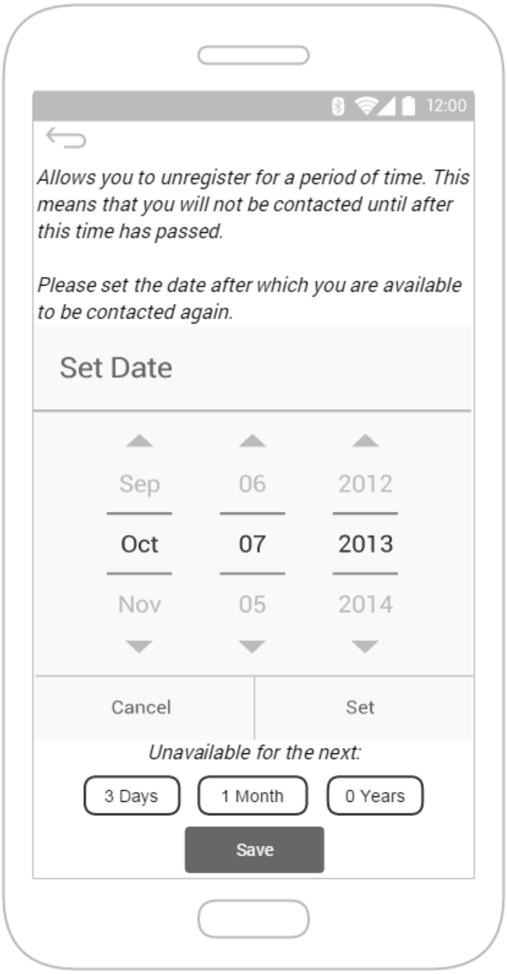
5.3.5.3 'Do Not Disturb' for a period of time We feel that the option to unregister from being contacted for a period of time is one which users would appreciate and use. Allowing the user to set a length of time in which they do not wish to be contacted will help to keep the retention rate of those using the app high. If this is not implemented then people may look to unregister from the service and then re-register later increasing the chance that some never re-register because it is too much hassle.

This feature could be used in a variety of situations such as a responder is going on holiday for a week or two, they are physically unable to help for a while (for example a broken foot) or other personal matters which mean they are not in a position to help. Whatever the reason our app will give the user the ability to specify a period of time that they are not to be contacted in.

5.3.5.3.1 App Design Our settings menu should clearly show a category for allowing the user to say they are unavailable for a time. From here they should be taken to a new settings pane in which they are able to set the amount of time they are unavailable for.



Menu describes each item below the button to change the settings for that item.



The settings pane allows the user to select a date until which they do not wish to be contacted.

This pane allows them to select a date until which they do not wish to be contacted, at the bottom the time until the selected date is displayed in days, months and years and there is a save button at the bottom. Once the save button is pressed the date is uploaded to our servers via the GCM upstream messages service and is then stored in our database.

5.3.5.3.2 Server Modifications In order to provide this feature our search algorithm will need to be modified to check if a do not contact date has been set. There are three possible genres of value for this field:

- Null (i.e. not set)
- A date in the future
- A date in the present or past

If an individual's records have no date set then the individual is available for contacting and the algorithm proceeds as normal. If there is a date present and it is in the future then this individual is ignored by the algorithm. If there is a date and it is in the present (current day) or the past then the algorithm should replace this value with null and then proceed with the individual as normal.

This feature will provide little extra overhead to the server as there will be only one extra comparison (between the saved and current date) which will be run if a user is selected as a candidate to be contacted.

5.3.5.3.3 Database Modifications This feature is tied to an individual rather than a device - if a responder selects a do not disturb date on one device, we want to ignore them completely (on all devices) until that date.

As a result, this feature is trivial from a data perspective: it is simply represented by a new `DoNotDisturbUntil` field in the `Responders` table. Minute precision is provided by SQL Servers `smalldatetime` type, and the column will default to `null`.

Responders

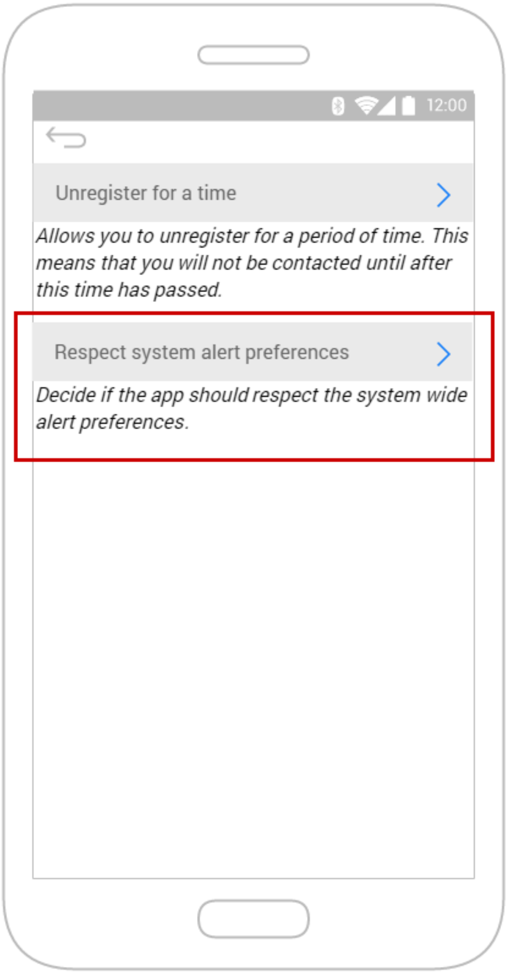
	Column Name	Data Type	Allow Nulls
🔑	ResponderID	int	☐
	Forename	nvarchar(128)	☐
	Surname	nvarchar(128)	☐
	Registered	datetime2(7)	☐
	DoNotDisturbUntil	smalldatetime	☑
			☐

5.3.5.4 Respecting system alert preferences We have decided to include an option in the setting menu which will allow the user to toggle whether the app should respect the operating systems alert preferences. This controls whether the app should be silenced by the devices silent or do not disturb modes.

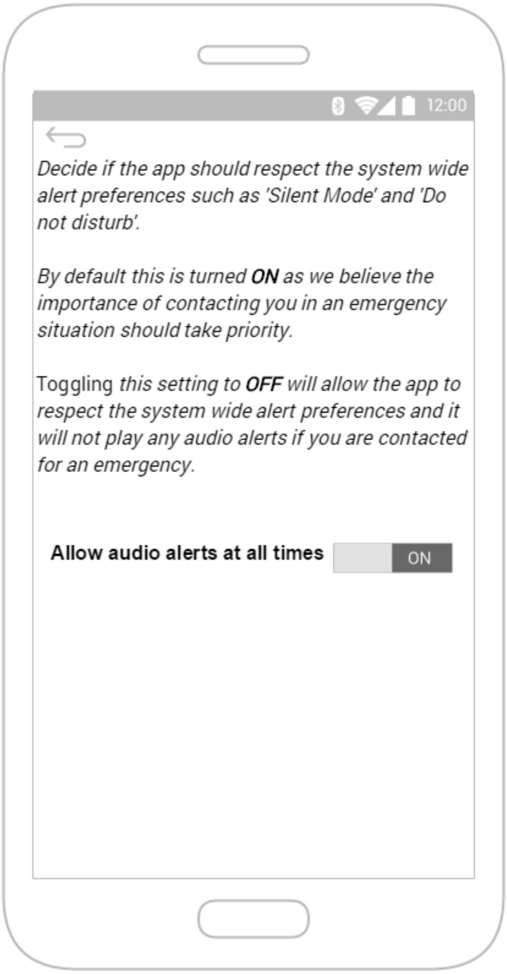
We have decided to include this toggle for situations where the user does not wish to unregister for a period of time, but does not wish for the alert to make a sound and feels they would be able to notice an alert regardless. However, we have decided that by default this feature is disabled so that the app will ignore the system alert preferences.

The reasoning for this stems from the nature of the app: in an emergency situation every second is vital and a user not realising they have a notification would mean that the search would pause for 30 seconds until the alert is automatically declined. We also believe that users are of a mind that the importance of potentially saving a life is above any inconvenience caused by the users phone going off. Also the frequency of messages from this app is likely to be quite low from an individual's perspective so it is unlikely that this will prove to be a common issue for them.

5.3.5.4.1 App design Our settings menu should clearly show a category for allowing the user to say they are unavailable for a time. From here they should be taken to a new settings pane in which they are able to set the amount of time they are unavailable for.



Menu describes each item below the button to change the settings for that item.



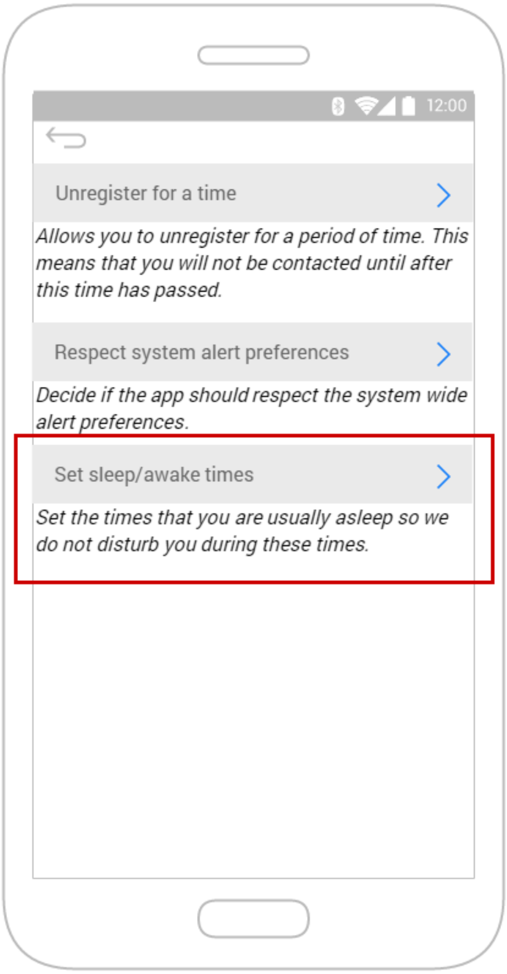
The settings pane allows the user to select if they want audio alerts at all times or not.

Here you can see that the effect of toggling this button is clearly defined to the user and it is explained that the default setting is that its turned on and why that is. Toggling this button will change how the notifications created when a new message is received from GCM are categorized.

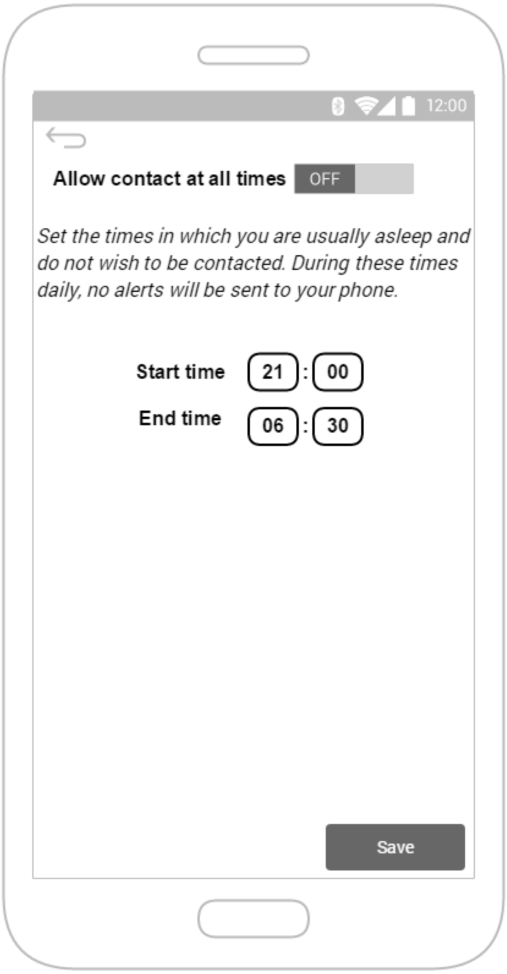
Different categories determine if the notification is affected by Androids Priority Mode and therefore if they are silenced. Setting the notification to an ALARM type [56] will mean the sound plays regardless of the system state. This settings value will need to be stored locally and then used when building the notifications for an alert.

5.3.5.5 Let the user specify the hours they will be asleep and thus unavailable This option allows the user to set a time when they are likely to be asleep and therefore do not wish to be contacted about any emergencies. There is also an option for users who do not mind being contacted at all hours. When toggled to ON the rest of the settings pane is greyed out and unavailable to change.

5.3.5.5.1 App Design The settings pane for this feature allows the user to enter a start and end time during which they are usually asleep. This will then be sent to our server when the save button is pressed and exclude this user between the two times. By default the app is set to allow contact at all times because we feel that it is best to let the user select the appropriate times for themselves. Also the frequency of messages from this app is likely to be quite low from an individual's perspective so it is unlikely that being woken in the night by this app will be a common problem for an individual.



Menu describes each item below the button to change the settings for that item.



The settings pane allows the user to set the times they are usually asleep.

When the save button is clicked the start and end date are both sent to our server via GCMs upstream service and stored on our database. If allow contact at all times is set to ON, the start and end times are set to midnight, which is interpreted by the server as no sleep period.

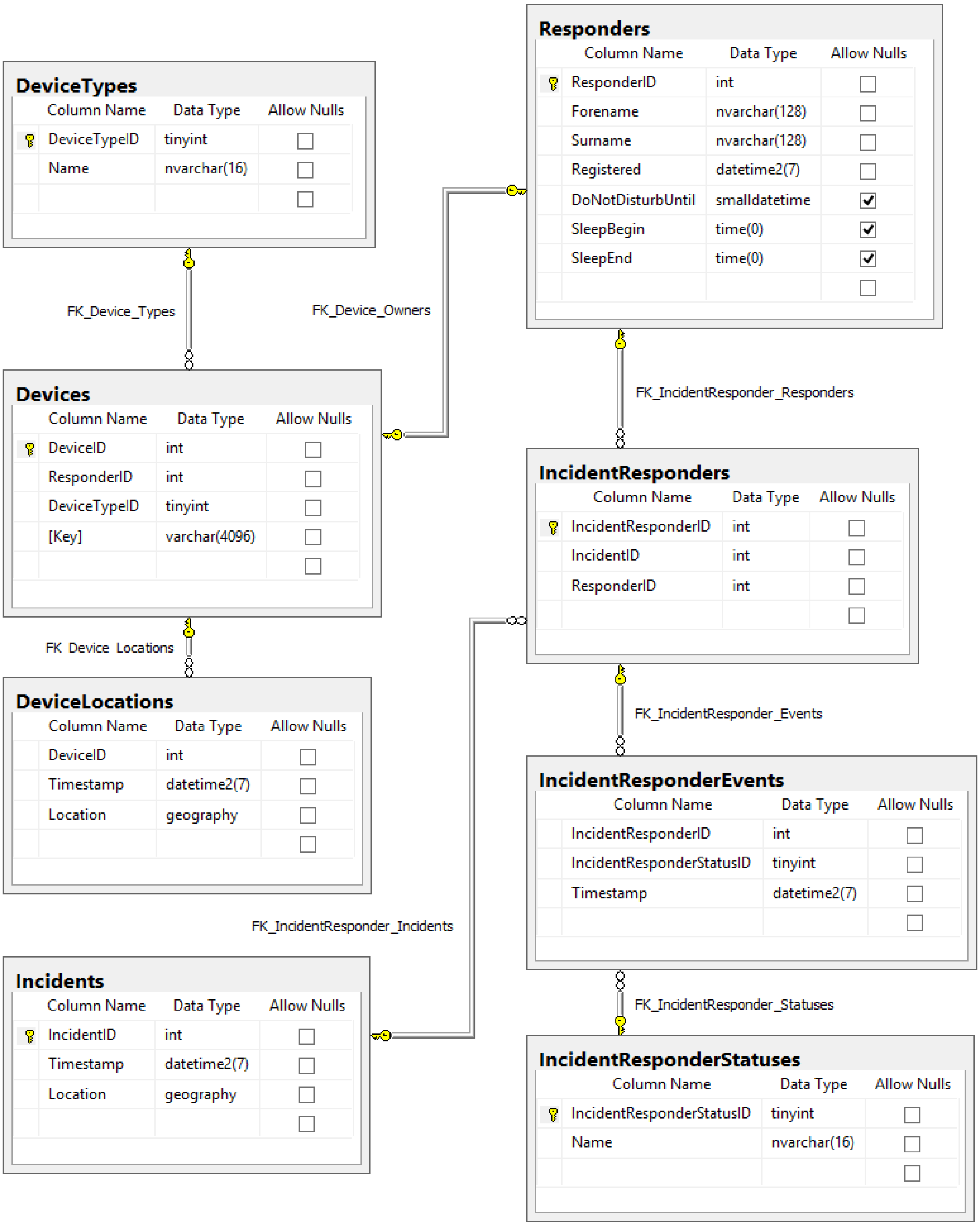
5.3.5.5.2 Server Modifications Our search algorithm will need to be modified so that once an individual is selected there is a check that the current time is not between the start and end of the sleep times. This feature will provide little extra overhead to the server as there will be only one extra comparison (between the sleep times and current time) which will only be run if an individual is selected as a candidate to be contacted.

5.3.5.5.3 Database Modifications Similarly to the do not disturb feature, the sleep times are tied to the responder rather than their individual devices, so the data is places in the `Responders` table. It will be contained in two fields, `SleepBegin` and `SleepEnd`, which will both default to `null`, indicating no times configured. Regarding the type, we only need accuracy to the minute, and no date component is required (as sleep reoccurs every night). SQL Server doesnt have a perfect match, however the closest is `time(0)` which can handle a seconds component as well - it is more accurate than required.

Responders

	Column Name	Data Type	Allow Nulls
🔑	ResponderID	int	☐
	Forename	nvarchar(128)	☐
	Surname	nvarchar(128)	☐
	Registered	datetime2(7)	☐
	DoNotDisturbUntil	smalldatetime	☑
	SleepBegin	time(0)	☑
	SleepEnd	time(0)	☑
			☐

This leaves the database design at the end of this iteration as follows:



5.3.6 Iteration 6 - Improving User Selection

5.3.6.1 Aim As it stands, users are selected based on their straight line distance from the location of the incident. This works at a basic level, but is suboptimal and problematic for several reasons.

5.3.6.2 Problems

5.3.6.2.1 Impassable Terrain Not all distances are equally quick to cover. A 200m journey along a flat road is relatively quick, whereas a shorter journey up a steep hill - or indeed over or up a cliff - poses more of a problem for responders. This will either substantially slow the responder down while they find a route, or mean they reject the request for assistance.

5.3.6.2.2 Mode of Transport Similarly the current selection mechanism doesnt take into account the rate at which responders can cover the distance to the emergency. Those in cars will usually move quicker, but traffic conditions will influence this, so it cannot be universally regarded as superior. In addition, the selection algorithm is only given positions of responders - it is unaware of their mode of transport, whether theyre moving towards or away from an incident etc.

5.3.6.2.3 Elevation This is a more subtle problem, but it could cause major delays in built up areas. The current algorithm works in two-dimensional space, assuming two equal distances are equally quick to traverse. As a result, cities containing lots of tall buildings with subtle access points pose a problem. Someone initially appearing to be closer may take longer to reach an emergency, as they have to go further away to reach an escalator or lift. This problem will also emerge whenever paths cross or overlap, such as when a road crosses a river, built over a path along the bank.

5.3.6.3 Solutions There are some solutions to all of the issues described above, however before they are considered, so must be performance. Currently, straight line distance is calculated by the database engine using the `STDistance` function. As a result, it is incredibly fast and so conceivable to run on the latest data from every responder for every incident. More involved solutions, like route-finding, cannot be done in this way. They require significant overhead and so the speed of operation must be considered.

The issue of impassable terrain can be overcome by using a mapping service (such as Google Maps or Ordinance Survey) to get details of terrain and routes. If the surface profile between the responder and the incident is relatively flat, the straight line distance becomes a more effective measure of travel time. Likewise, if there is a path or road that starts and ends near the points, it suggests they are reasonably accessible to each other.

The next step up from this is doing a full search between each responder and the incident. While this would give the most accurate result, and could even be used to guide responders to incidents, it is also very inefficient, even when highly parallelised. As a result, it would have to be used with another technique, possibly straight line distance. In this case, straight line distance would be used to filter out those who are simply too far away to arrive within the 8 minute window. Using the average walking speed of 3.1 mph [61], and assuming this is the slowest any responder will travel at (a safe approximation, given most responders will walk briskly or run), the maximum straight line distance any potential responder should be from an incident is 665m. Therefore, `STDistance` can be used to select all responders within this radius of an incident, and then do run a search algorithm on them. If this proves to be too much in congested areas, even after optimising performance (in theory every responders search could be run in parallel), the limit would have to be lowered.

However, it is important to realise that there are no universal solutions to the issues described above. It is impossible to keep a constant record of whether a responder is on foot or bicycle, in a car or on a bus that is travelling a set route - which may happen to deviate away from the incident in the future. It would be foolish to select someone based on a footpath between them and the incident, when they are in a car.

After a fair amount of thought and brainstorming, it became evident that the best overall solution is to use the historical data available to make better decisions. Peoples lives are normally full of patterns, which can be exploited to our advantage. This, combined with using the last few locations of a person to calculate their speed - and therefore likely mode of transport (with on foot being a good assumption if no movement), means it should be possible to give accurate predictions of where each person will be at a given time. Using this data, the system will be able to answer questions like Who will be nearest to this incident within the next minute?, compensating for travel distance as the time increases. It will be a compromise between sending someone further away sooner, and telling someone currently further away who we believe is travelling towards the emergency.

Of course, in practice the system would always choose someone closer to the incident now, as it is safer, however specifying such a system is beyond the scope of this project. Although a very interesting mini-project, a large testing effort would be required to design it, so our specification will remain with eliminating those too far away using straight line distance, before running search on the remaining ones using walking as the mode of transport.

---

6 Project Management

---

7 Personal Reflections