Software Design Study

Final Report

Emergency Systems Team

George Brighton (1220556), Matthew Flint (1247903),
Deyan Genovski (1259269), Martin Mihov (1229174), Robert Zlatarski (1256494)
{gxb256, mxf203, dag269, msm274, rxz294}@bham.ac.uk

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.

May 1, 2015

Contents

1 Introduction

1.1 Background

When someone dials 999, the call is taken by an emergency operator at an Operator Assistance Centre (OAC), which are run by the phone companies (e.g. BT). They then route the call to an Emergency Control Centre (ECC), of which there are several for each emergency service. Any call involving police (regardless of other services requested) is routed to one of their control centres. The police will make a record of the call, before passing it over to any other involved service, e.g. London Ambulance Service for giving medical advice. London Ambulance Service and the Metropolitan Police's systems are integrated, so passing calls between them is easy, however if the incident requires London Fire Brigade, the message is routed to the Contact Desk, where someone has to pick up a phone and ring them on a dedicated line.

1.2 Current Problem

Our initial research into the area led us to the conclusion that the current software used by the emergency services was not up to scratch. Poor software, namely CAD, should have no place in 2015 and we believe that this issue needed addressing.

As it turns out so did the government. We discovered an ongoing deployment of CommandPoint (developed by Northrop Grumman) to replace the old system. This rather ruined our plans for a project as we did not see the need to design something that was already in existence. Instead, we tried to find issues with CommandPoint and work on solutions to those, however this fell short when we discovered that the design of CommandPoint is very comprehensive and we could only find minor issues with it.

This lead us on to the final direction of our project, we decided to develop some innovative extensions that could be implemented along side CommandPoint which would provide a host of useful features to the emergency call takers and the general public.

1.3 Currently Available Systems

1.3.1 Call Handling System (CHS)

CAD has a text-based interface, so about 5 years ago a system called CHS (Call Handling System) was brought in. It effectively adds a more user-friendly interface to CAD, with demand created using it converted into CAD messages. Demand can be created using both CAD and CHS, but new employees are only trained to use CHS. Whenever CHS crashes, new call takers have to write out demands on paper and pass them to the rare, remaining CAD-trained person to type into the system.

1.3.2 Computer Aided Dispatch (CAD)

CAD is the current dispatch system used by the Metropolitan Police Service. Developed by Unisys Corporation in the 1980s and originally based off a baggage handling system, it has long surpassed its predicted lifespan, and many contracts have been created to extend support for the system. It is slowly being phased out and replaced by Northrop Grummans CommandPoint system.

1.3.3 CAD Backup Facility (CBUF)

CBUF is a mirror CAD system, used if the latter crashes, runs slow or undergoes maintenance. The problem is switching between the two - demands trickle over over the period of about an hour. This is a serious problem if something serious comes in and one system immediately crashes - the switch to the other system is almost immediate, but it can take an hour before the last events from the old system appear. If CAD is going down for planned maintenance, supervisors in the despatch print their open incident lists so they at least have a clue about what was outstanding. The problem is when maintenance ends, the trickling problem happens again.

Whenever possible, the Met uses CAD over CBUF, presumably because the maintenance contract specifies CAD should be fully functional most of the time and CBUF less so, and it would waste money to aim for full functionality of both systems all the time.

1.3.4 Emergency Help via SMS

In case of an emergency, people have to call 999 and request help. Unfortunately, people with hearing loss or speech impairment cannot use their mobile phones for voice calls. Currently, there exists an Emergency SMS service [49], in order to give these people the opportunity to request help in the case of an emergency.

The current system, however, is very outdated, not very reliable and difficult to use. Some of its problems are there because it is based on SMS technology. Although sending an SMS requires a very low network coverage, it is very unreliable. It can take from a few seconds to a few minutes to deliver a single SMS message [50]. Apart from that, the order in which multiple SMS messages are delivered, may differ from the order in which they are sent.

Because of this, the process of requesting help in the case of an emergency through SMS has become rather difficult to use. It consists of 4 steps [51]:

1. Registering the phone - This is required in order for the emergency center to filter real emergency messages from the fake ones. The user has to send an SMS with the text "register" to 999. Then an SMS, containing terms and conditions will be received, to which the user has to reply, using a second SMS with the text "yes". At the end, the user will receive a confirmation SMS, indicating that the current phone number is registered.
2. Write the emergency SMS - The user has to write as much of the required information as he can in a single SMS, in order to give details about the location and the incident and minimise the need of further communication through SMS with the call center.
3. Send the SMS to 999 - Sending it as a single SMS makes sure that the information comes in the desired order and can be handled properly by the operators.
4. Wait for response - A response will then arrive either asking for more information, or informing that the emergency team is on the way. Unfortunately, the "SMS Delivery Report", which some operators provide, does not guarantee that the SMS was received by the emergency center. The Emergency SMS service suggests that response is usually received within 2 minutes, but if a response is not received within 3 minutes, a new SMS should be sent.

2 System Overview

2.1 Emergency Services App

The Emergency Services App is aimed towards all members of the public with access to a smartphone. A companion for the traditional 999 call procedure, it adds a multitude of features and extensions which benefit both user and operator.

2.2 CPR App

The CPR App is aimed towards people who are already trained in CPR and first aid. The app is intended to be used passively, installed once and forgotten about, until it detects that there is an emergency situation nearby that the individual could help at. The idea is that an individual trained in first aid can be called upon to act as a first responder until the emergency services arrive.

2.3 Separation

Initially, we were designing our system as one app containing both of the above features, however we soon realised in the design stages that this was impractical. Parts of one feature were starting to encroach on areas of the other and we realised the two use cases lend themselves better to separate apps, as they offer fundamentally separate services. It is also true that users may only want one of the two use feature sets, for example if the apps were combined, everyone would be carrying around the ability to be contacted via the CPR system. This is impractical and a waste of storage on users' devices.

We believe that separating the features into separate apps will not inconvenience the majority of users. Those who wish to participate in the first responders initiative need only download a small app which they can set up once and 'forget about' until they are contacted with an emergency near by.

2.4 Servers

Our apps connect to our own remote servers which are used as the main 'hub' for all communication between the client devices and the emergency dispatchers. In some cases, connections to these servers are direct and in others we are using the Google Cloud Messaging service to efficiently send data back and forth.

2.5 API

We have developed an API alongside our apps so that external companies can easily integrate our products into their existing systems. We decided to go along this route as we do not know enough about the potential systems our system could be installed in to be able to integrate into them ourselves. An API also allows our services to be implemented in other systems that we might not have considered during the planning stages of our project, making it more accessible.

3 Modules

3.1 Emergency App

3.1.1 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].

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

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 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.

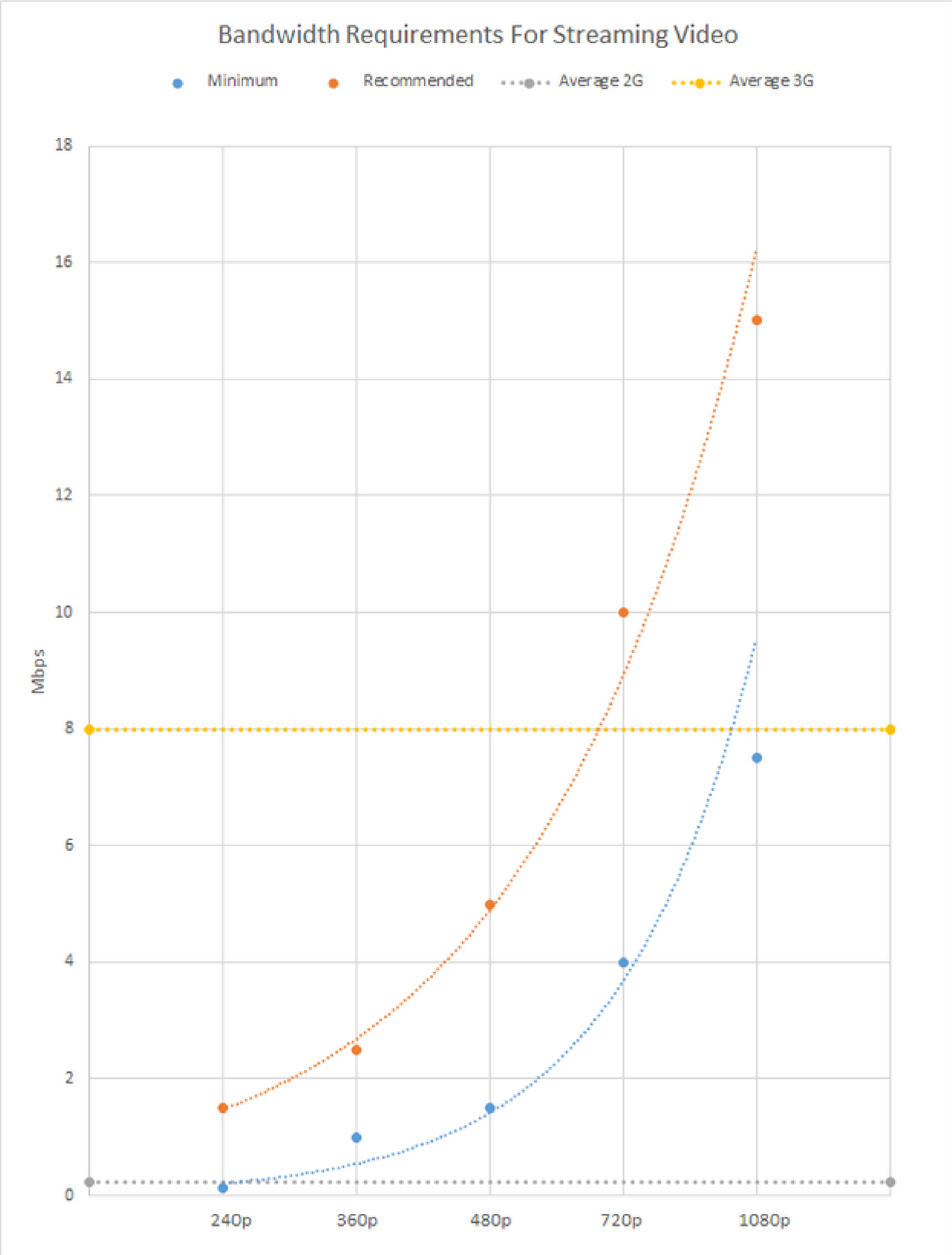


Figure 1: The bandwidth requirements for video streaming.

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.

3.1.2 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.

3.1.3 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.

3.2 CPR System

3.2.1 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.

3.2.2 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.

3.3 API

As it was previously explained the project consists of several systems communicating and interacting with one another. The communication between different systems is done via an Application Programming Interface (API). Since the current systems declined to share their API, we have decided to create and provide an API to third party developers. They should be able to integrate it seamlessly in their products.

It was decided that all interactions with the system, where possible, will be made through the API. This includes the communication between the backend user interface and the Server infrastructure as well as the communication with the mobile applications, where possible. Using a well documented API throughout the project ensures that all functional capabilities are exposed. This in turn means that if the existing emergency systems decide to integrate with our product the integration could be done without a modification of the existing infrastructure.

When researching different API technologies a number of requirements had to be taken into account.

1. *The API needs to be platform independent*
   Since the the system being developed has Android application, Server infrastructure, Operator front end, one of main requirements of the API is to be cross-platform.

2. *The technology should be popular*
   Given the variety of the systems involved, the technology used to develop the API should be well established so that properly developed and tested libraries can be used. Since popular technologies are well known to developers, this manner of implementation will ease third-party integration with the platform.

4 Specification

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

4.1 Common Technologies

4.1.1 Google Cloud Messaging

4.1.1.1 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.

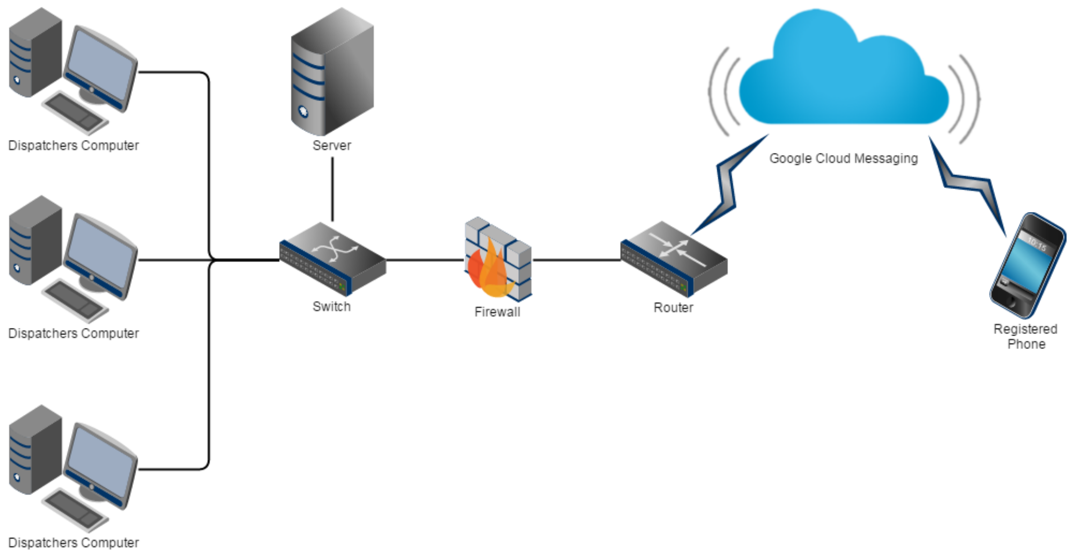


Figure 2: Network map of how our system will communicate between devices and the server

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.

4.1.1.2 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].

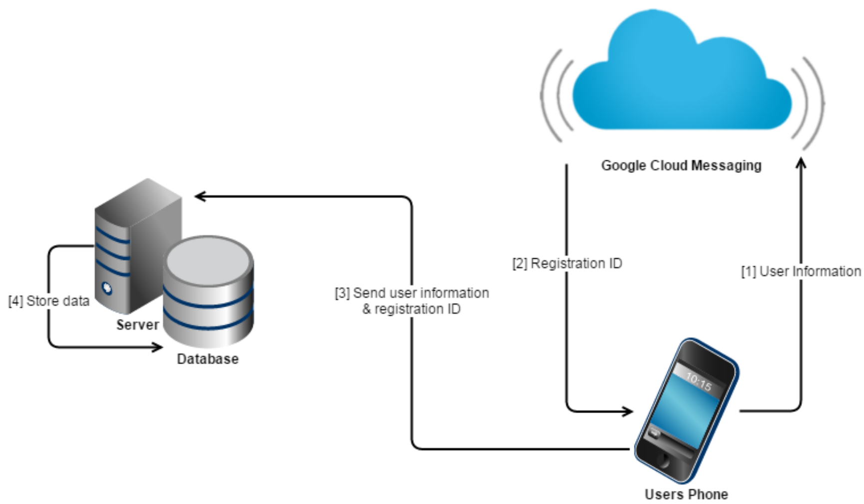


Figure 3: Diagram of the initial setup and registration with Google Cloud Messaging

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.

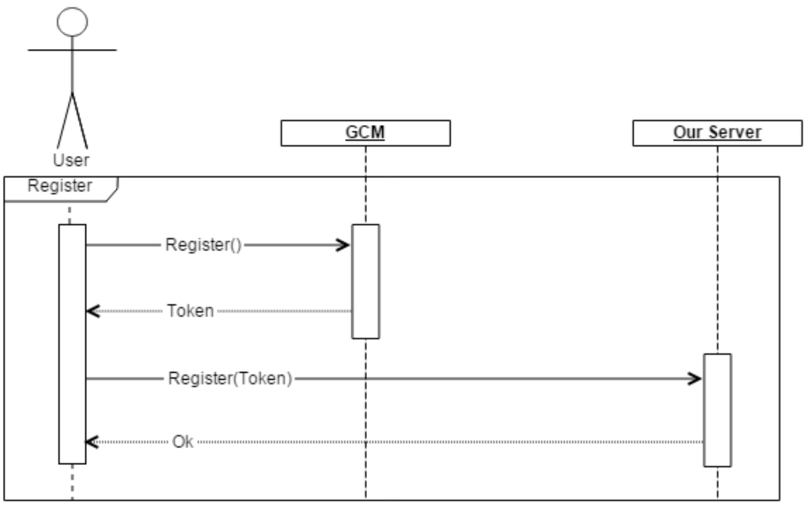


Figure 4: Sequence diagram showing interactions when registering a device

4.1.1.3 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 Google's Messaging Server which then sends it directly to the device associated with that ID.

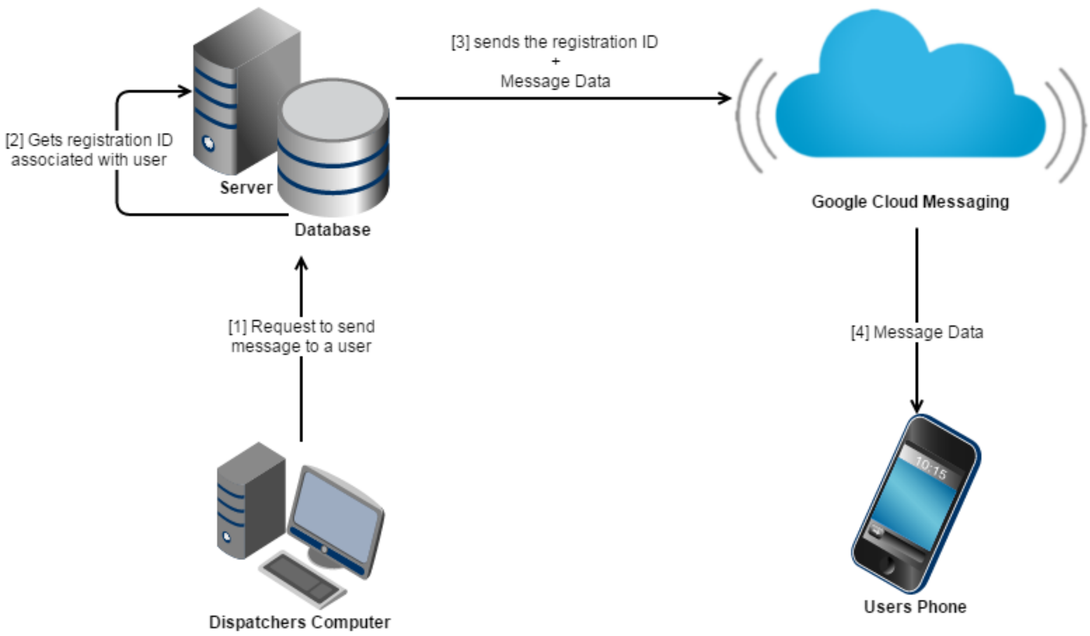


Figure 5: Diagram illustrating sending of a message to a clients device

We have decided to use the XMPP protocol as the connection between our server and GCM's 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.

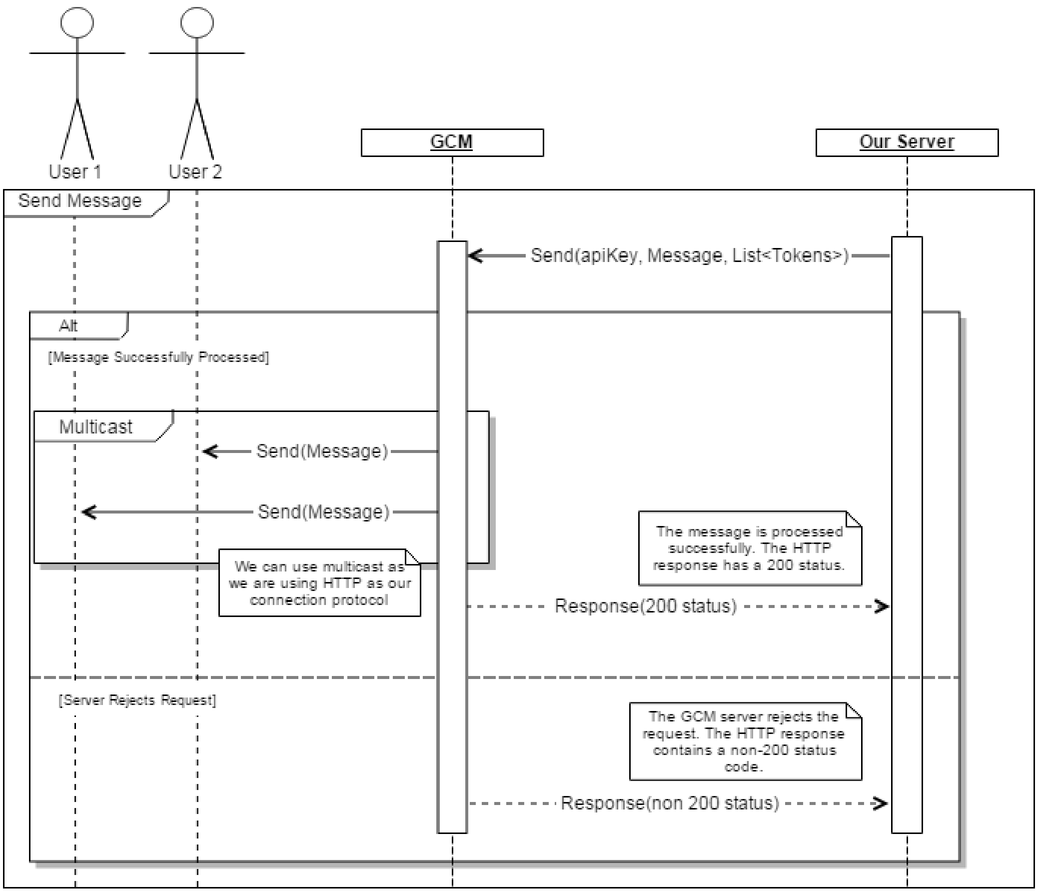


Figure 6: Sequence diagram showing the interactions when sending a message to the client device

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].

4.1.1.4 Sending Upstream Messages We can also use GCM to send upstream messages from the clients device to the server via Google's CCM (Cloud Connection Service). This works by embedding the server's 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>
```

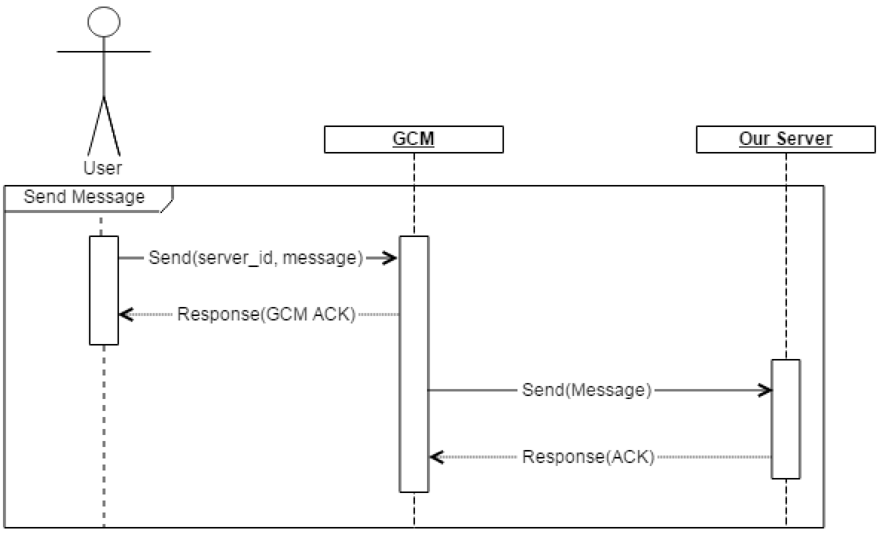


Figure 8: Sequence diagram showing the interactions when sending a message to the server from a client device

4.1.2 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: 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 isn't 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.

4.2 Emergency App - Video Streaming

4.2.1 User Stories

Iteration 1:

1. As a user I want to be able to start a video conversation with the emergency operator in order to show what is happening.
2. As an operator I want to be able to prompt the user to start a video call.
3. As a user I do not want the video call to interrupt the call with the emergency center.
4. As a user I want to be able to stop the video without interrupting the call with the emergency center.

Iteration 2:

1. As a user I want to set the preferred stream quality.
2. As a user I want the application to set the quality of the video based on my internet connection.
3. As a user I want to be able to choose whether to use the rear, front or both cameras of my phone during the call.
4. As a user I want to be able to use the flashlight of the phone during the call.

Iteration 3:

1. As an operator I want to be able to review old video records.
2. As an operator I want to be able to skip/seek to a certain time of the video footage and return to the live scene.

Iteration 4:

1. As a user in the case of an unexpected crash or an unintentional close of the application, I want to be able to continue the live stream when I start the app again.
2. As an operator I want to be able to go back to the current open stream if the UI unexpectedly crashes or closes.
3. As an operator in the case of a crash in the backend UI I want to be able to review the recording of the stream, once the UI has restarted.

Iteration 5:

1. As an operator I want to be able to see other video streams, if they are streaming the same incident.

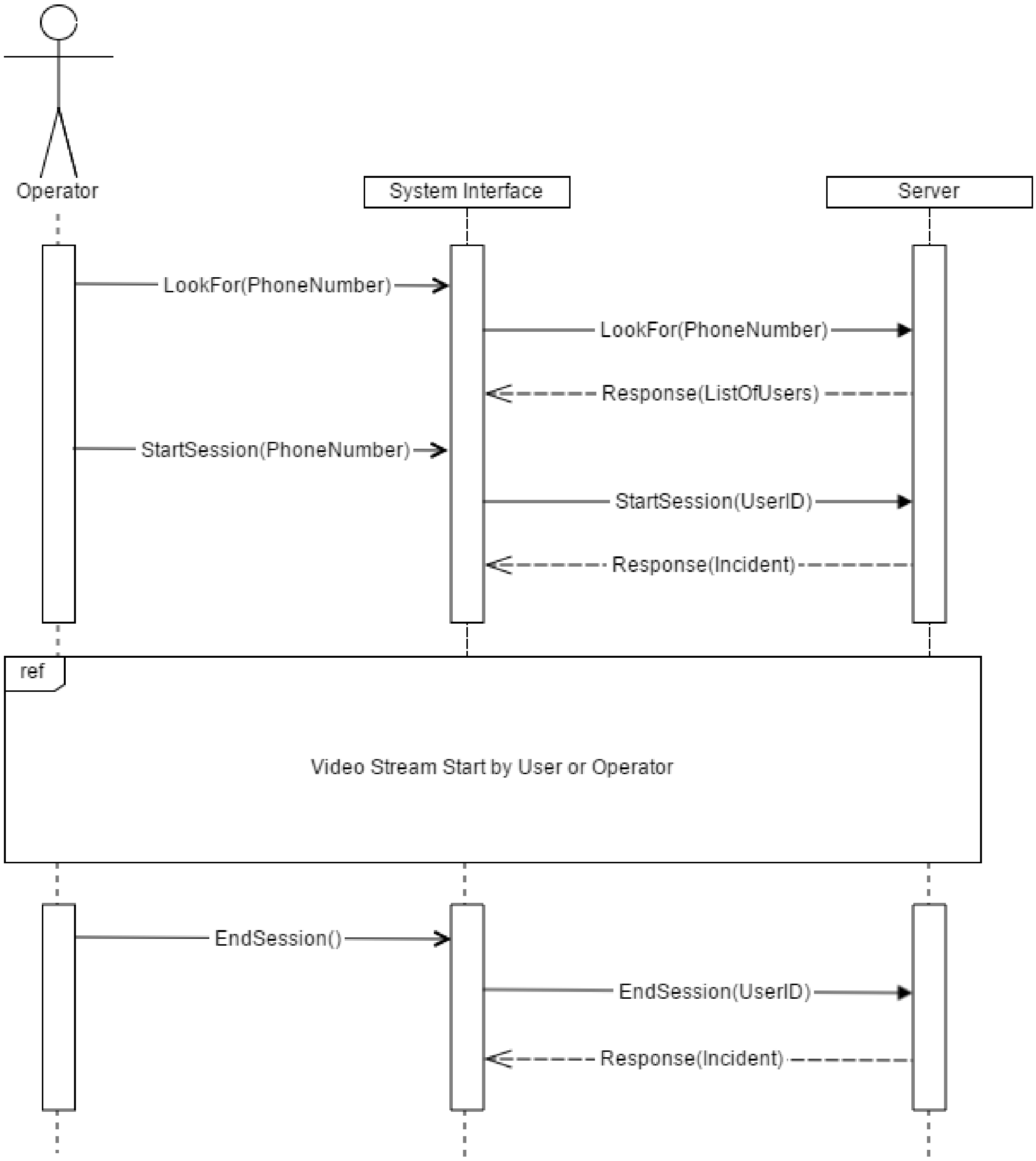
4.2.2 Iteration 1 - Basic Functionality

4.2.2.1 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.

4.2.2.2 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.

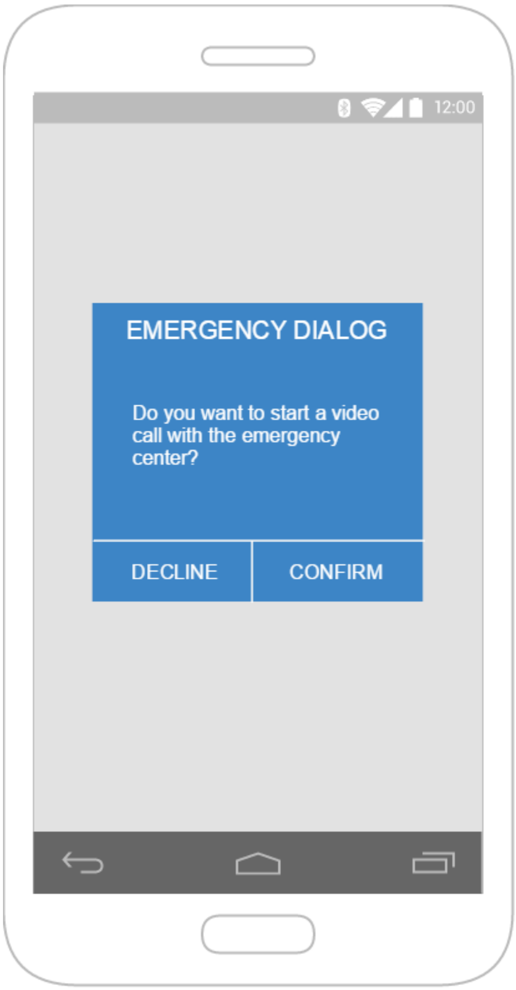


Sequence diagram 1. The operator creates a session.

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.

4.2.2.3 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.

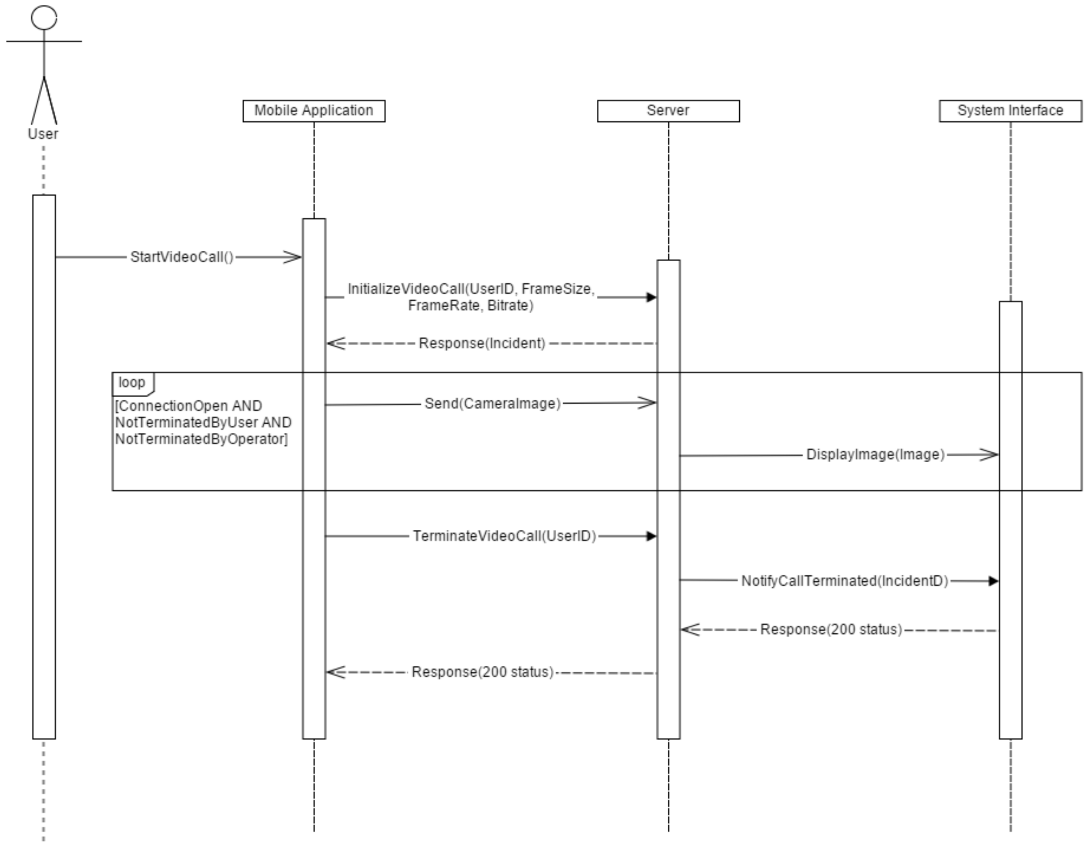
```
sips:<token>@<server_url>:<server_port>
```

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.

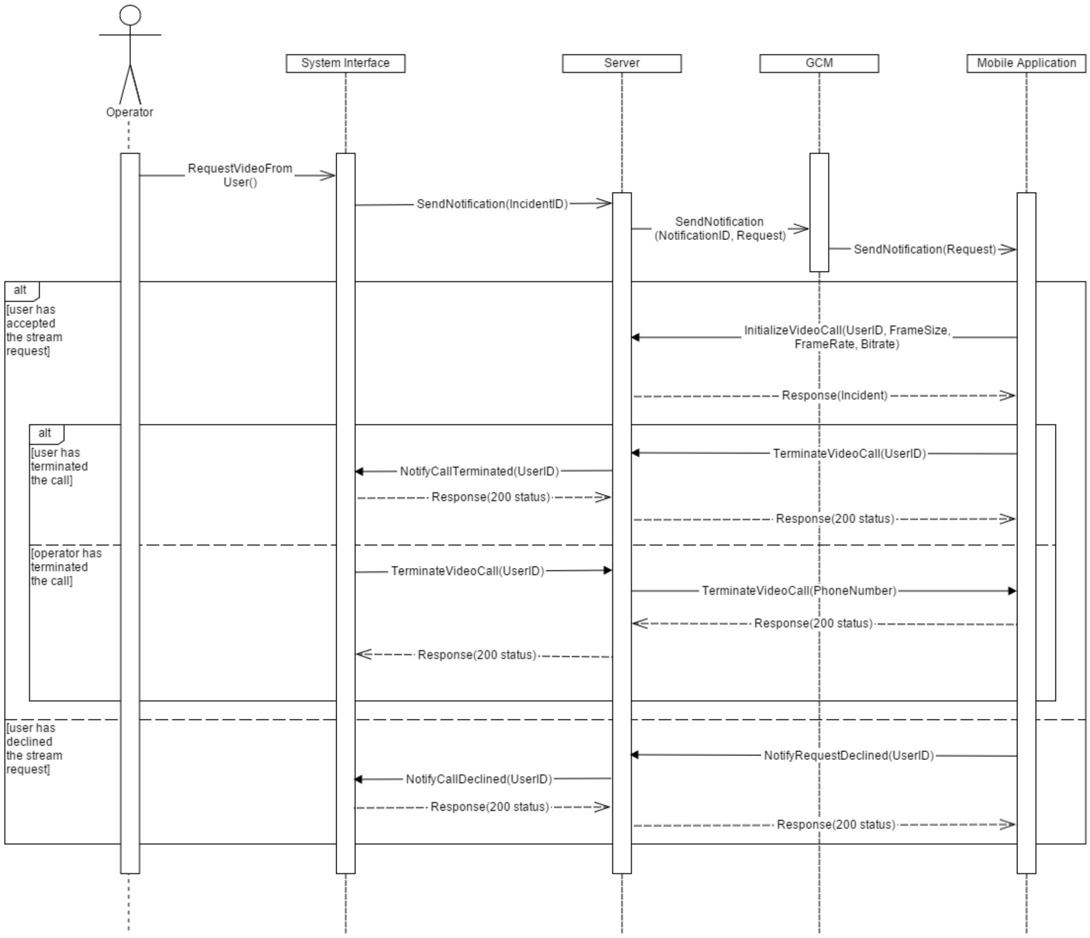
Then the current settings for the stream and camera are sent over: frame size, frame rate and bitrate. Once the socket is opened and configured, the camera is started and streaming begins. The stream is encoded with VP9. We have chosen to use VP9 as a video compression technology over one of the most 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.

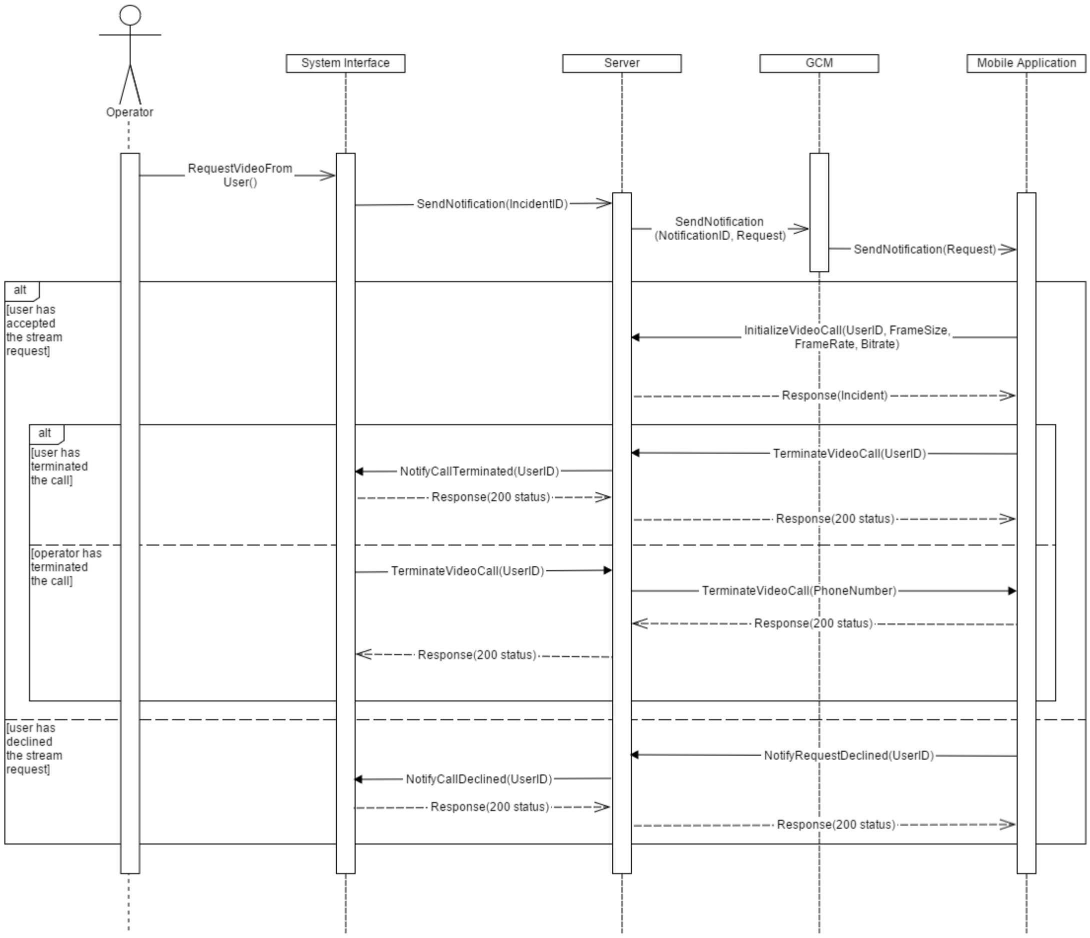
The user and the operator are able to stop the video stream at any time, using the on-screen navigation on the app or the system interface, without interrupting the call itself.



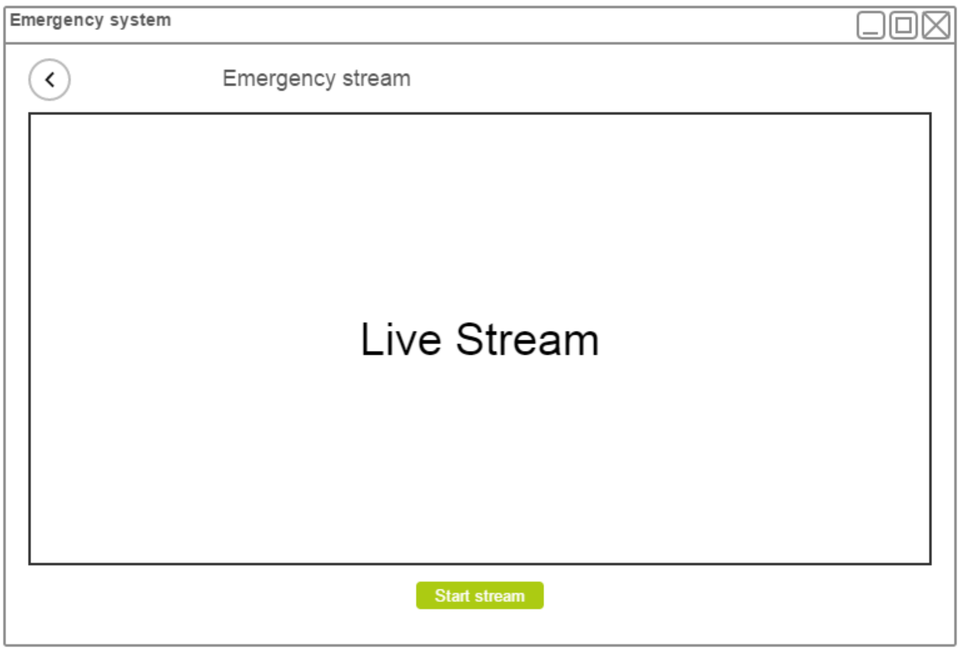
Sequence diagram 2. The user starts a video stream.



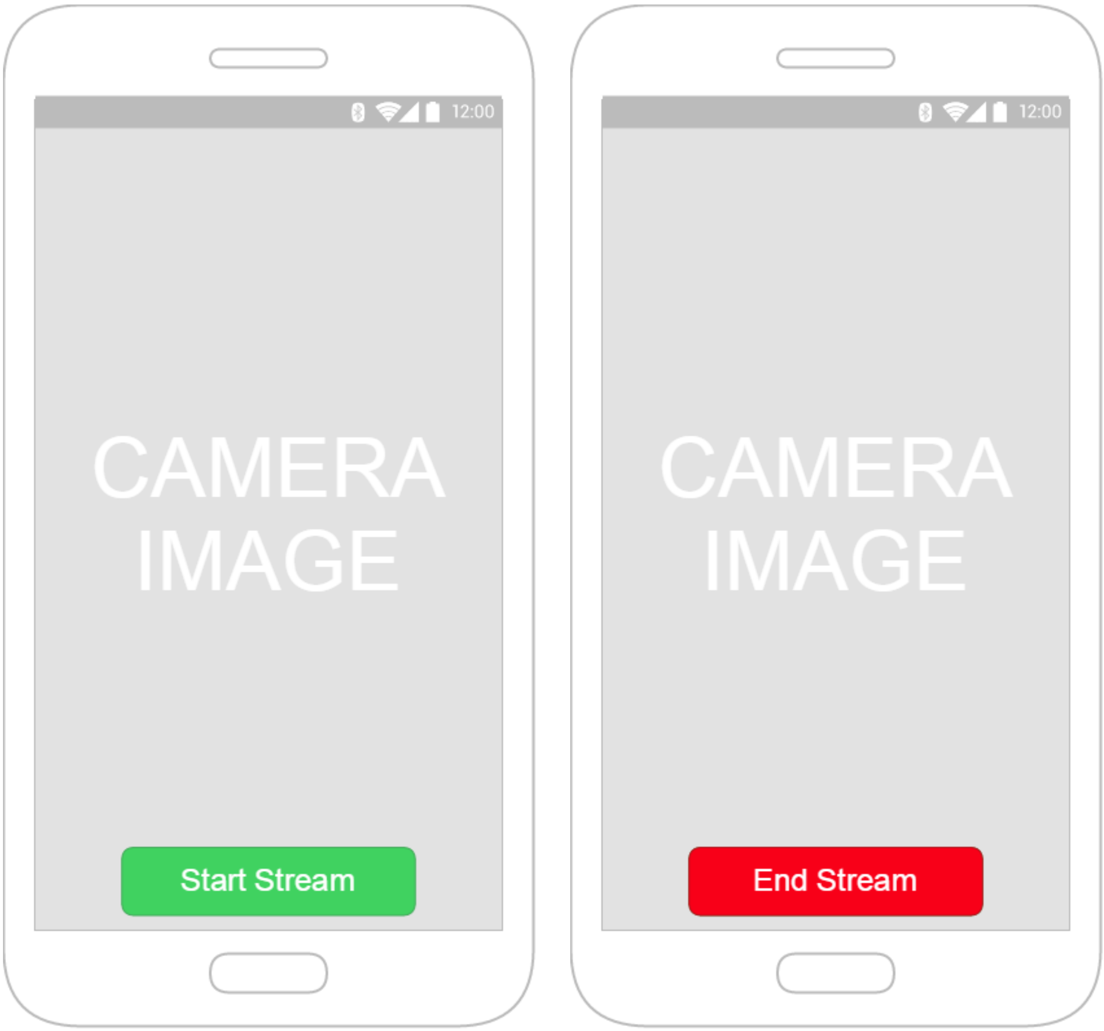
Sequence diagram 3. The operator requests the start of a video call.



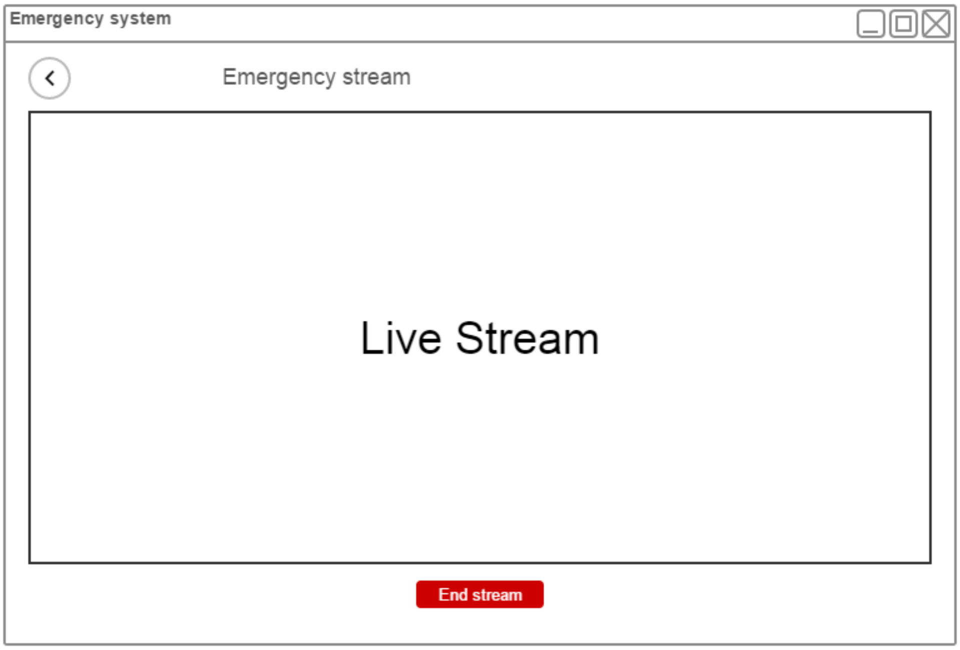
Sequence diagram 3. The operator requests the start of a video call.



Backend screen to request a video stream



Screens to manage the start and stop of the video stream by the user



Backend screen to stop the stream

4.2.3 Iteration 2 - Enhanced User Experience

4.2.3.1 Aims This iteration aims at enhancing the user experience during a video conversation. It includes setting up video stream quality and introducing video stream interface buttons.

4.2.3.2 Setting the quality of the video stream The user is able to change the prefered stream quality both in the application settings, when not in a call, and through the live stream interface, during a call. If the settings are not set manually, the quality is set automatically based on the mobile phones capabilities and the internet connection available.

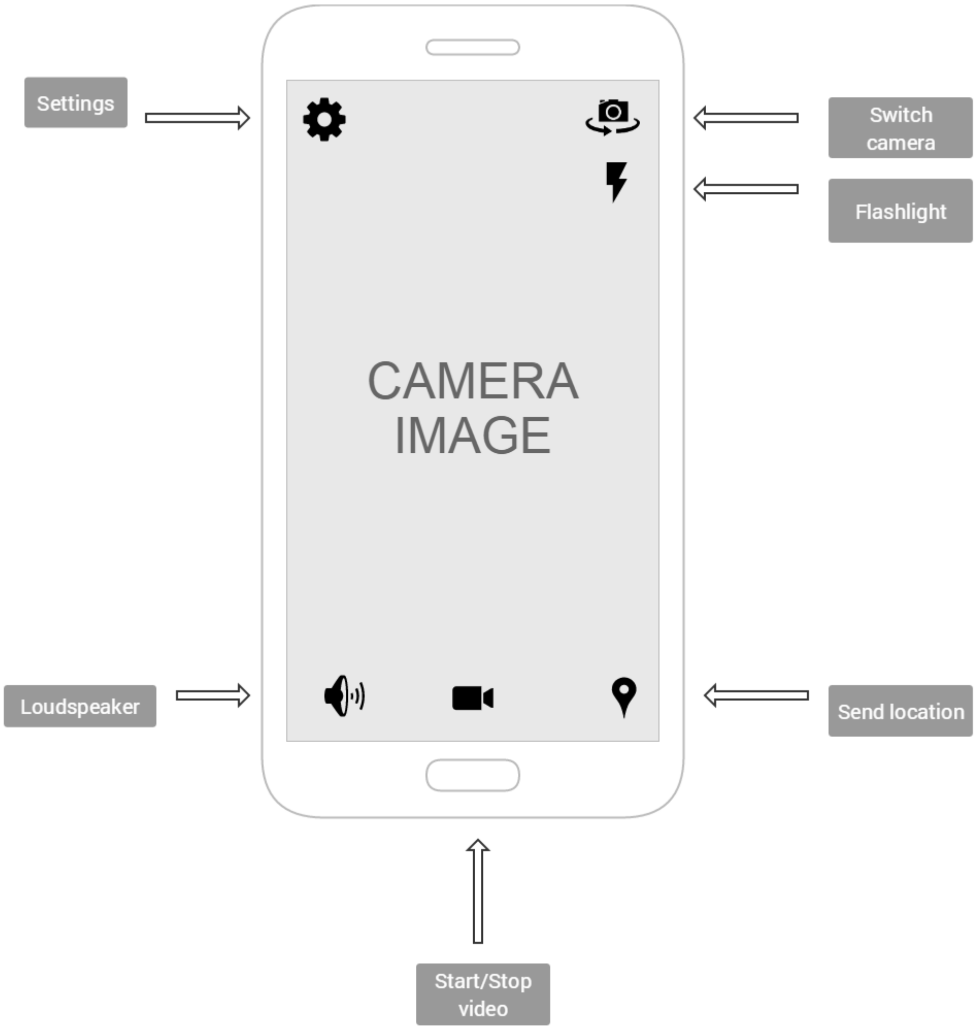
The user will be able to choose between the following stream qualities: low, medium and high. Each of those will be set with unique frame rate, bitrate and resolution in order to optimize the performance and quality of the stream. A typical example of the importance of this feature is when a user is having a video stream and his internet connection is not very good, it would be reasonable that he changes the quality to medium or low, in order to keep the connection stable. This will also allow the user to minimize the used bandwidth, by choosing a lower quality setting.

High quality will mean encoding the video with a resolution of 720p (1280x720) [45] [39]. The video bit rates will be a maximum of 4000 kbps to a minimum of 1000 kbps, where the default will be 2500 kbps. The frames per second will be adjusted based on the devices camera capabilities, but never exceeding 30 fps [46]. Medium quality option will provide a resolution of 360p (640x360) [45]. The video bit rates will be a maximum of 2000 kbps to a minimum of 500 kbps, where the default will be 1000 kbps. Low quality option will provide a resolution of 240p (320x180) [45]. The video bit rates will be a maximum of 1000 kbps to a minimum of 400 kbps, where the default will be 750 kbps. The video bitrates for the different quality options will be adjusted automatically by the application, based on the users internet connection.

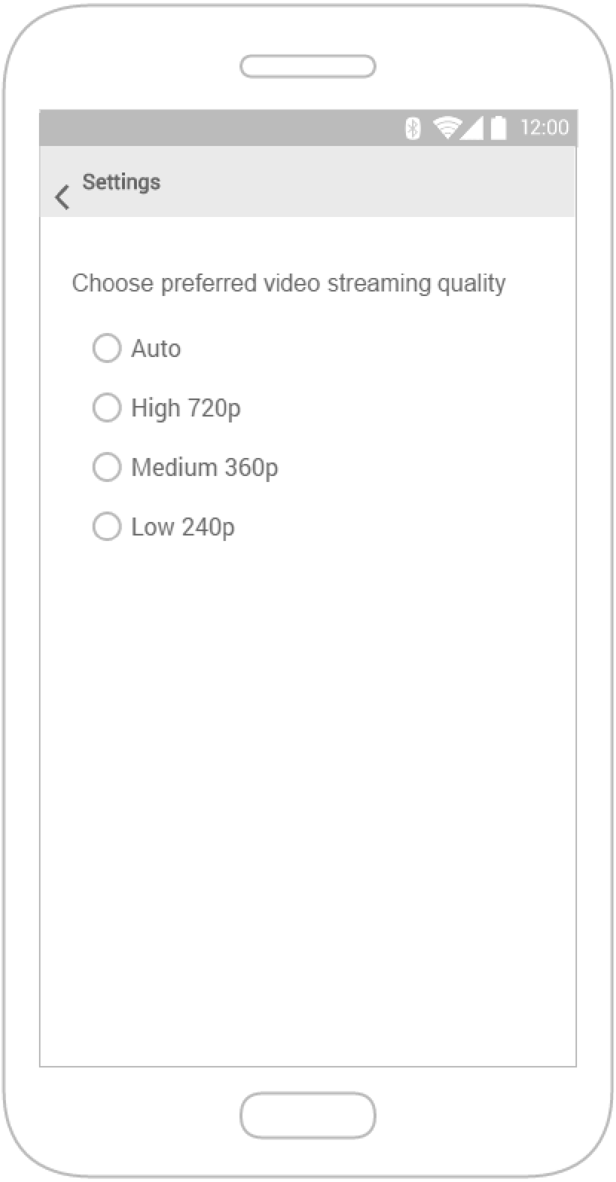
4.2.3.3 Video Stream UI Apart from choosing the quality of the video stream, the user will be able to choose which camera of the phone to use, during the stream. The user interface during a video stream will allow the user to easily switch between cameras. Selecting a different camera will also affect the stream quality, as usually both cameras are not the same. If the users phone does not have a front camera, the button will not be available.

Another feature that the on-screen UI will provide, is switching the mobile phones flashlight on and off, in order for the user to be able to stream video in low light conditions. If the users phone does not have a flashlight, the flashlight button will not appear on his screen. In order to make the call still available, while the user is streaming video and is holding his phone away from his ear, a button for enabling and disabling the loudspeaker has been included too.

All icons of the buttons included in the on-screen UI are the default icons used in many already existing applications that are using the camera. This way, they will be intuitive to the users and wont need further description, which is an important point in the UI. If there are too many on-screen buttons, the picture wont be clearly visible and the user may experience difficulties during the stream.



On screen UI during a stream



Stream quality settings

4.2.4 Iteration 3 - Enhanced Dispatcher Experience

4.2.4.1 Aims The aims of this iteration is to provide the functionality of skipping and seeking through a live video stream and being able to review old videos.

4.2.4.2 Video storage and skip/seek to a certain time As some emergency scenes may need further attention, all video streams will need to be stored.

In order to do this, a storage server will be needed, that is separate from the one running the backend UI or the database server. It will also need to be extendible, in order to include more storage devices or replace old ones.

The video stream will be saved immediately after the server receives it from the client and will continue writing to the same file, until the stream has been stopped. This will ensure that all streams are always saved and that if the backend UI crashes, the stream will still be saved and accessible. Each saved video stream will be labeled by its date and time, as well as with the ID of the operator taking the call. An example stream started on 21/04/2015 at 13:11 by an operator with ID 123456789, would look like:

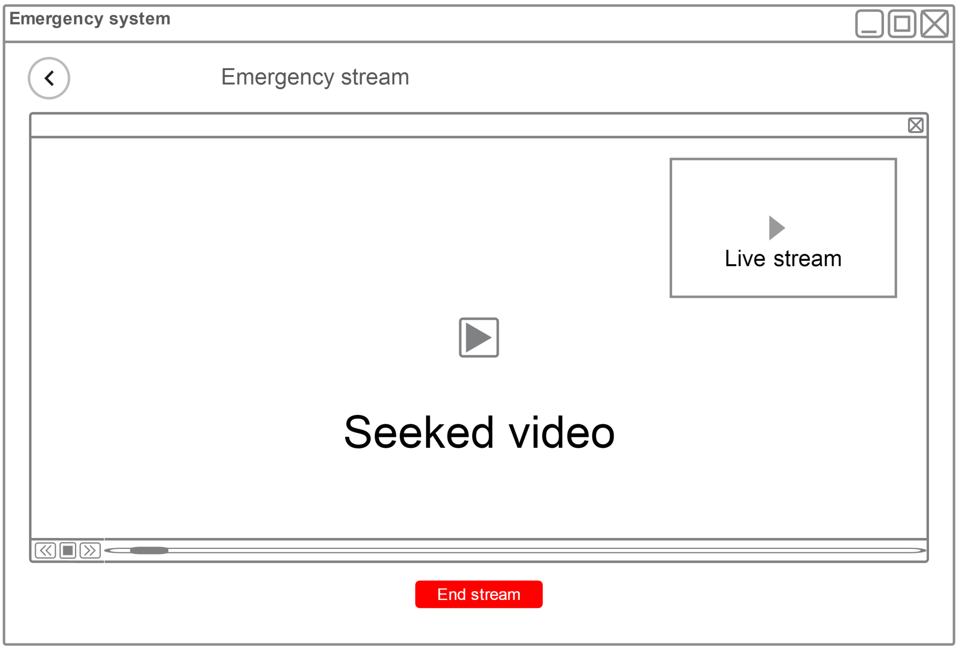
`2015-04-21-13-11-123456789.webm`

The WebM file format is a container for VP9 video streams and will be used when storing the video streams [47] [48].

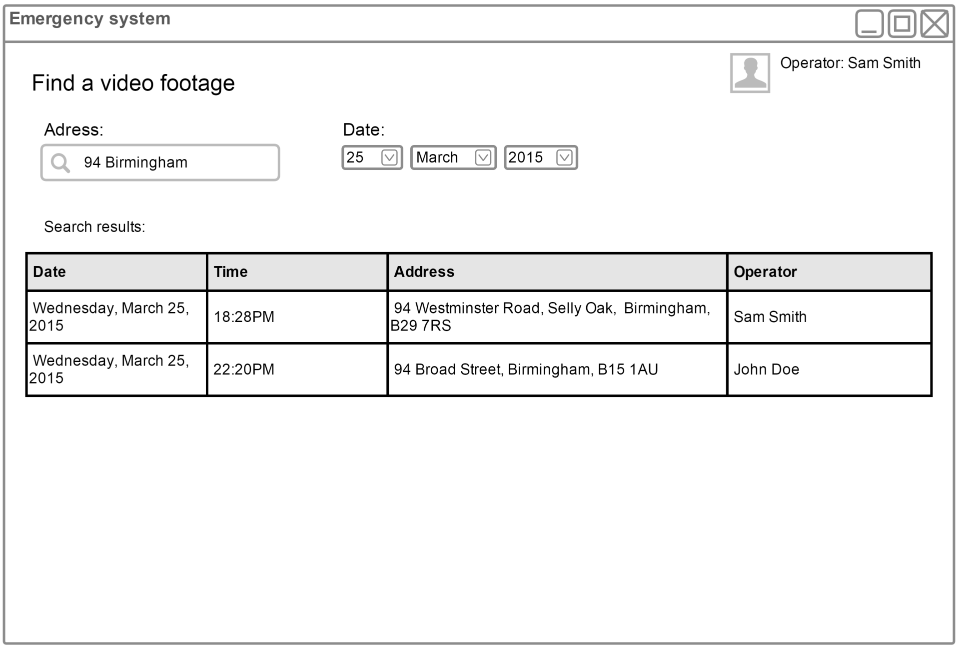
The name of the file will be saved in the corresponding incident in the database and it will be searchable in our systems user interface.

Moreover, even during the live video stream, operators will be able to skip/seek through the already recorded video and return to live footage at any time they want. This could be very beneficial, when something needs to be examined in depth, while the video stream is still in progress. When the operator chooses to go to a certain time of the stream, while it is still active, the live video will continue to be shown in a box in the top right corner of the main video window, while the scene that is chosen starts in the main one. This way the operator can see what is being streamed live and return to it immediately if necessary.

Licensed systems users will have access to old recorded streams for further investigation. Reviewing can be done either by playing the file directly from the storage device with a third party software, or by using the user interface to search for it and play it.



Seeking back to a past moment of the stream



Accessing old emergency sessions

4.2.5 Iteration 4 - Improving Reliability

4.2.5.1 Aims This iteration aims at improving the reliability of the backend and the mobile application.

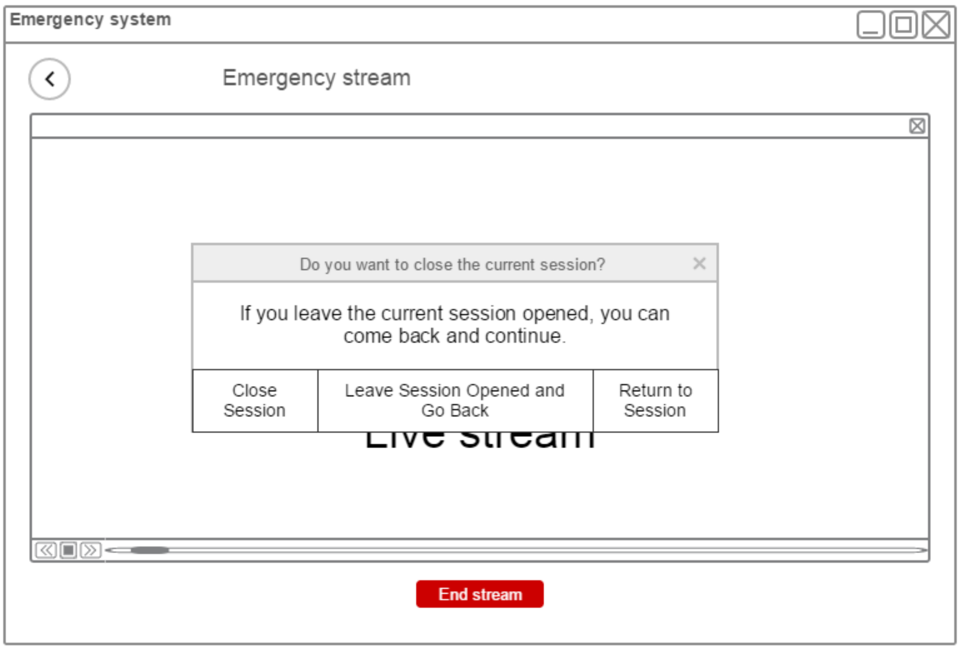
4.2.5.2 Restore on unexpected close There are events when computers may fail and the backend can close unexpectedly, closing the current emergency window. Such events include power cut, software bug or any hardware failure. As other actions can be taken in order to minimise the possibility of such event happening, the software by itself has to be able to react to it too. To do this, the system will adopt some new features.

Apart from that, the mobile application may face similar issues and restoring to the current emergency session, when the application has been closed or has crashed is an important feature.

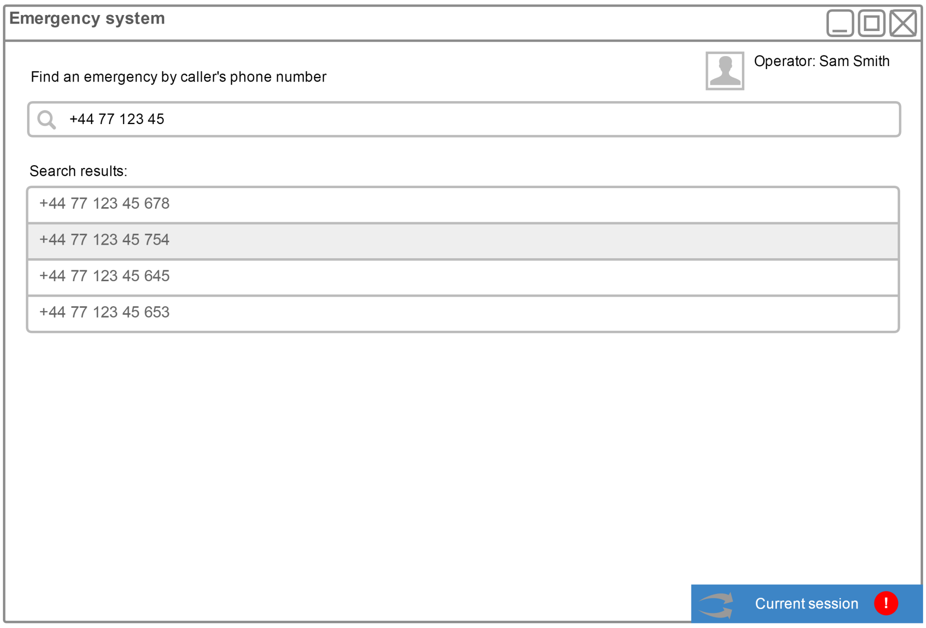
Each operator will be able to open only one emergency session at a time. He will not be able to open or start another session, until he has closed the current one. In order to provide this functionality, the operator will need to explicitly state that he wants to close the emergency session. This will be done by a Dialog, which pops up when the back button has been clicked during the stream. The dialog will provide 3 options: Close the current emergency session, Go back and leave the current session opened, Stay on the page and close the dialog.

In the main screen, the operator will be able to see if there is a current active session and will be able to open it. This will help reopen the session in the case of a crash of the program or when the operator closed the windows, leaving the session open. The user can continue streaming during this time, as the server will continue processing the data. After opening the session again, the operator will be able to skip/seek back and see what has happened during the time his stream window was closed.

If the mobile application of the user crashes, on restart it will automatically try to reopen the stream.



Confirmation of closing the session or navigating back



Notification of a current session

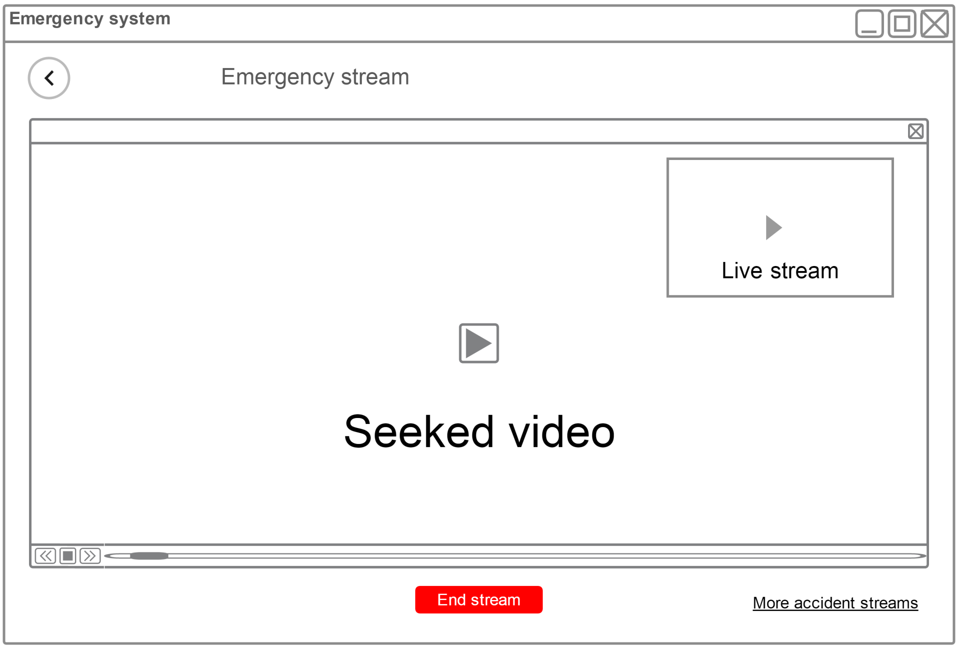
4.2.6 Iteration 5 - Multiple Live Streams

4.2.6.1 Aims Give the operator the opportunity to switch between multiple live streams, that are showing the same emergency. This will allow for the operator to assess the severity of the incident even better and he will be able to react properly.

4.2.6.2 Multiple Streams When multiple operators take calls, there might be the case when two calls are actually about the same incident. As there is a way of marking two calls as the same incident in the current systems, we decided to implement a way of combining the information from different streams.

When an operator has opened a stream, he will be able to see whether there are other streams for the same incident and will be able to click them and review them. He will be able to watch the live stream, as well as skip/seek to a certain part of the stream. This is possible as all the streams are being saved on the storage server and if two streams are being marked as one incident, they will become accessible by the operators handling the calls.

The backend user interface will allow to chose different streams and watch them, while the current live stream will remain as a thumbnail and the operator will be able to return to it with only one click. Every alternative stream will be marked by the unique telephone number that the user is calling from.



Confirmation of closing the session or navigating back

Notification of a current session

4.3 Emergency App - Chat

4.3.1 User Stories

Iteration 1:

1. As a user I want to be able to start a live chat with an emergency operator.
2. As an operator I want to be notified when a user starts a new chat session.

Iteration 2:

1. As a user I do not want to start a chat by an accidental pressing the chat button.
2. As a user I want to be able to quickly switch between video and chat windows.

4.3.2 Iteration 1 - Basic Functionality

4.3.2.1 Aims This iteration develops the core functionality of starting a live chat with an emergency operator. It is intended to provide the user with alternative ways of holding a conversation, when he is in an emergency situation.

At the most basic level, a user should be able to begin an online chat conversation with an emergency operator. Moreover, the operator should be instantly notified when a chat session has begun.

4.3.2.2 Starting a live chat with an emergency operator Once a users device has been registered, a chat conversation with an emergency team can be started. A user has to simply start the application and choose the emergency chat option. This will transfer him to a chat window, where he can start typing and send messages to 999.

In order to establish the connection between the server and the application, a WebSocket [54] will be opened between them. Through it, messages can be sent over in real time. It is designed on top of TCP, which provides reliability for delivering the messages and ensures that they arrive in the correct order. This is very important for the emergency chat, as notifying the user if a message has been delivered, can greatly increase reaction time. The WebSocket will be opened through the following URI:

```
wss://<server-address>:<port>
```

This opens an encrypted connection between the client and the server. Once a connection has been established, the user has to identify himself. To do this, the application sends the id received when registering during the first start of the application. The server will respond with the ID of the emergency session created for the current emergency chat. All messages will be sent in the JSON format and will contain authorization token.

Request from the application to start an emergency chat:

```
{
	"Type":"start_session"
}
```

Response from the server:

```
{
	"IncidentID":"abcd1234567890", -- Can be null if initiating a session was
	    unsuccessful
	"Type":"session",
	"Status":"open/failed"
}
```

After the session has been initiated, the user can start typing the emergency messages. Once the user starts typing a message, a notification is being sent, that the current Emergency chat is active and requiring operators attention. This is done in order to shorten the response time, by notifying the call center of new emergency chats, before the actual message has been received.

Application notifies the server for an active chat:

```
{
    "IncidentID":"abcd1234567890",
    "Type":"session",
    "Status":"active"
}
```

The information can then be sent in multiple messages, sending each bit of information as a separate message. The example given on the Emergency SMS website [51] can be broken down into the following messages:

Original: 'ambulance. man having a heart attack. outside post office. valley road watford'

Chat messages:

1. ambulance
2. man having heart attack
3. valley road watford
4. outside post office

This will allow for the operator to respond faster. When the message requesting an ambulance has been received, the operator can react accordingly and record the emergency in the system. When the description and the location has been received, the operator can quickly contact the emergency team in the corresponding region or ask for more information in just a few seconds. A simple chat message will also provide delivery report, as well as an indicator if the operator has opened the chat or not, which will allow the user to get faster information on whether the messages were delivered. If the message fails to be delivered, the application will retry to send it automatically, without the need of the user typing the message again. However, if the message fails to be accepted by the server for any other reason, a notification will appear on the users screen, describing the error.

Simple outgoing chat message (both from operator or from user):

```
{
        "IncidentID":"session1234567890",
        "Type":"message",
        "Message":"need ambulance",
        "Timestamp":"437647234",
        "TempMessageID":"temp1234567890"
}
```

Delivery confirmation message (sent to the sender of the message):

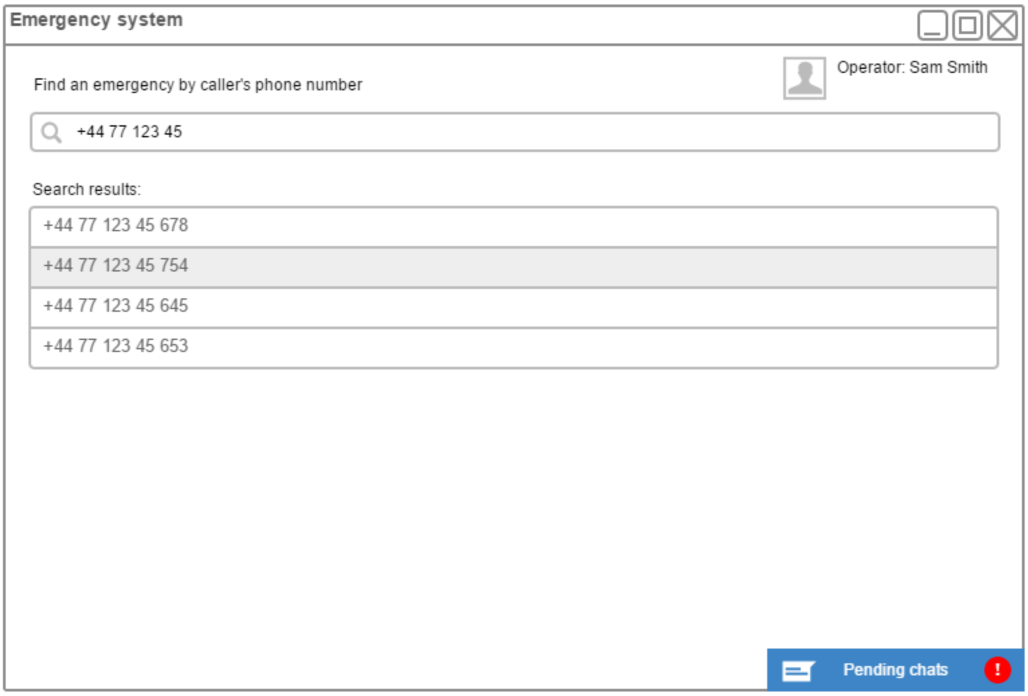
```
{
        "IncidentID":"session1234567890",
        "Type":"message_report",
        "TempMessageID":"temp1234567890",
        "MessageID":"message1234567890",
        "Status":"delivered/failed",
        "Error":"A description, which will be displayed in a popup." -- If message was
            delivered, this field is empty
}
```

Operator connected message (sent to user, once an operator opens the emergency chat session:

```
{
        "IncidentID":"session1234567890",
        "Type":"message_report",
        "MessageID":"message1234567890",
        "Status":"seen"
}
```

The user will be able to close the application at any one moment, however, the emergency session will be active until the operator decides to close it. If the user closes the application and tries to start a new chat session, while his current session is active, he will send the messages to the same operator and the same emergency session. This will handle accidental closes or crashes of the applications and will allow the user to go back to the conversation.

4.3.2.3 Sending a notification for a started chat session On the operator side, when the user selects chat option and starts typing, there is a notification that pops up that informs the operator that there is an emergency going on and that there are incoming messages. These notifications are acquired by getting all incident sessions with a status of pending chat.



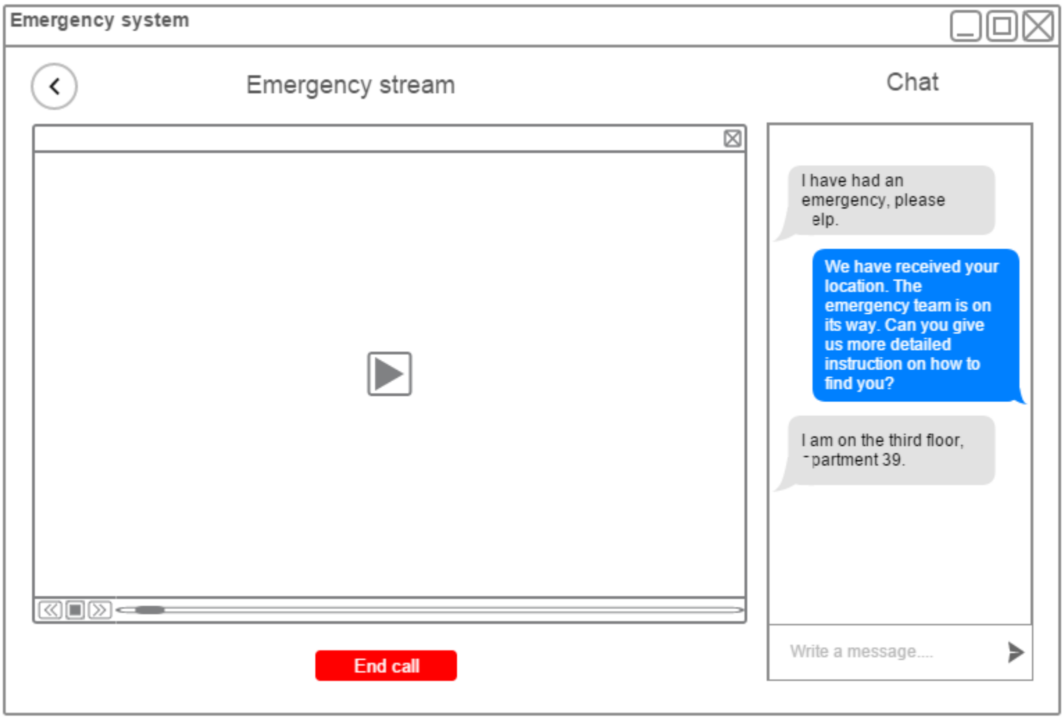
Backend notifies of new pending chats

When the operator selects to respond a chat, he is assigned an emergency session. In order to avoid multiple operators responding to one chat, transactions are used. When the operator responds to a chat, there is a transaction that is being sent to the database, that locks the session only for that operator. Transactions are the way to ensure that the users of the system are working with consistent information and provide error recovery mechanisms. The distribution works the following way:

1. The user sends a message.
2. The message travels to the emergency center.
3. The systems identifies available operators and notifies them that they can start an emergency chat.
4. The operator selects to start a chat.

5. A chat session starts and becomes private only to that operator, while it is being handled.
6. When the emergency has been handled and the operator closes the session, the user will not be able to send any more messages in the current session.

The backend will allow the operator to simultaneously use both the video stream service and the chat service, which will ensure that no time is wasted for switching between screens.



Both video streaming and live chat will be available to the operator in the same window

4.3.3 Iteration 2 - Security and Accessibility

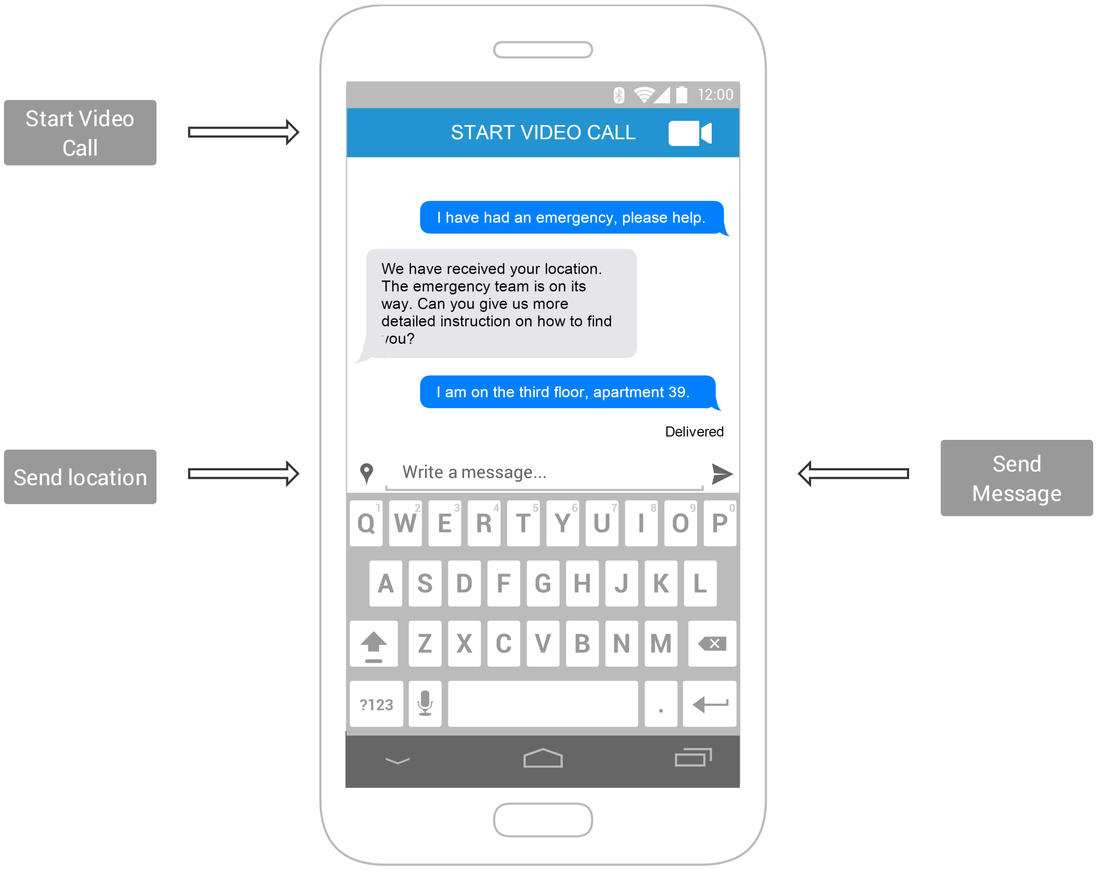
4.3.3.1 Aims This iteration aims at adding security and accessibility features, as well as making the user interface more intuitive, by adding quick buttons for easier access to the other features of the application.

4.3.3.2 Preventing chat button miss-click In order to prevent unintentional opening of the emergency chat or chat button missclick, a confirmation dialog will appear, informing the user of any further consequences. Under UK law [55] a hoax caller is a person who for the purposes of causing annoyance, inconvenience or needless anxiety to another, sends, or causes to be sent, by means of a public electronic communications network, a message that the person knows to be false. It is a criminal offence to make a hoax call. Thus, criminal proceedings can be brought against people who use the emergency chat for reporting false incidents or just to abuse the caller by any means. If the user confirms the dialog, he will be able to proceed to the chat dialog and will be able to start immediately an emergency chat.

4.3.3.3 Enhancing user interface As one emergency session will provide access for the current user to both video stream and emergency chat, the navigation in the mobile application between the chat and the video stream should be easy and intuitive. To do this, new buttons were introduced both in the chat UI and the video stream UI. During a chat, the user will have a big button on top of the screen, which will directly start a video stream. Apart from that, there will be a quick button for sending the users location in the chat screen. This will allow the user to send his location, without the need to type it.

The button icon, that is included in the video stream screen and gives quick access to the emergency chat screen, also contains an indicator. It shows whether and how many new messages have been received. So a user may quickly see that a new message has arrived and navigate to the chat screen.

Navigating from the chat screen to the video stream will keep the chat open, however, navigating from the video stream to the chat screen will stop the stream, as the user will have no indication that he is streaming video to the emergency center.


Start Video Call
Send location
Send Message
12:00
START VIDEO CALL
I have had an emergency, please help.
We have received your location. The emergency team is on its way. Can you give us more detailed instruction on how to find you?
I am on the third floor, apartment 39.
Delivered
Write a message...

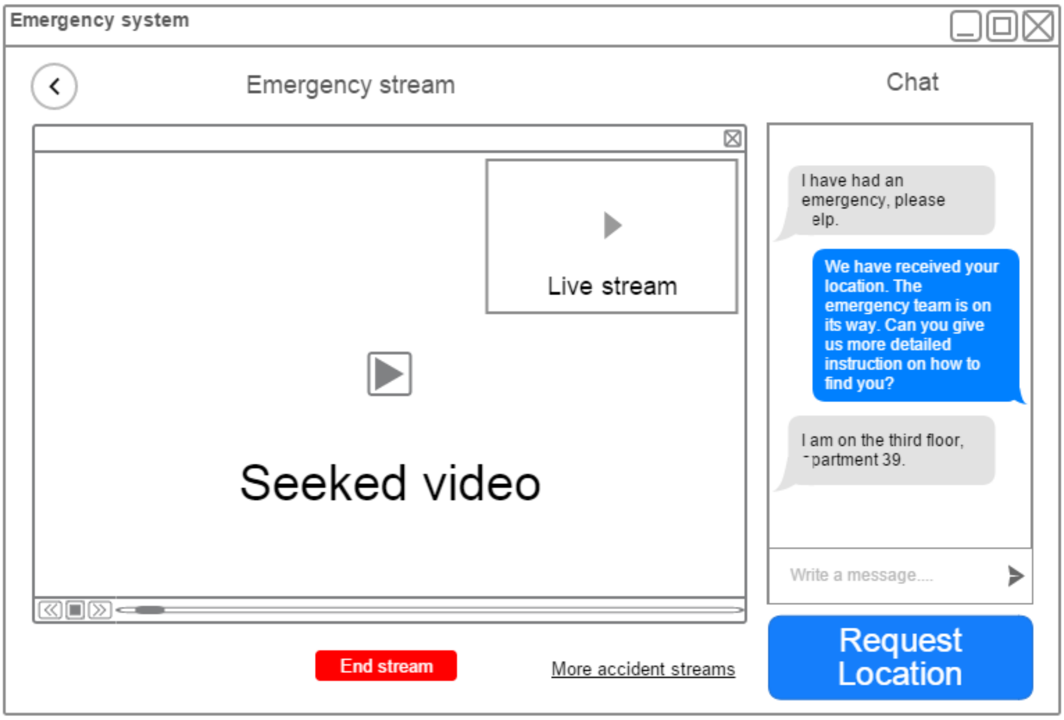
4.4 Emergency App - Automatic Location Sending

4.4.1 Iteration 1 - Basic Functionality

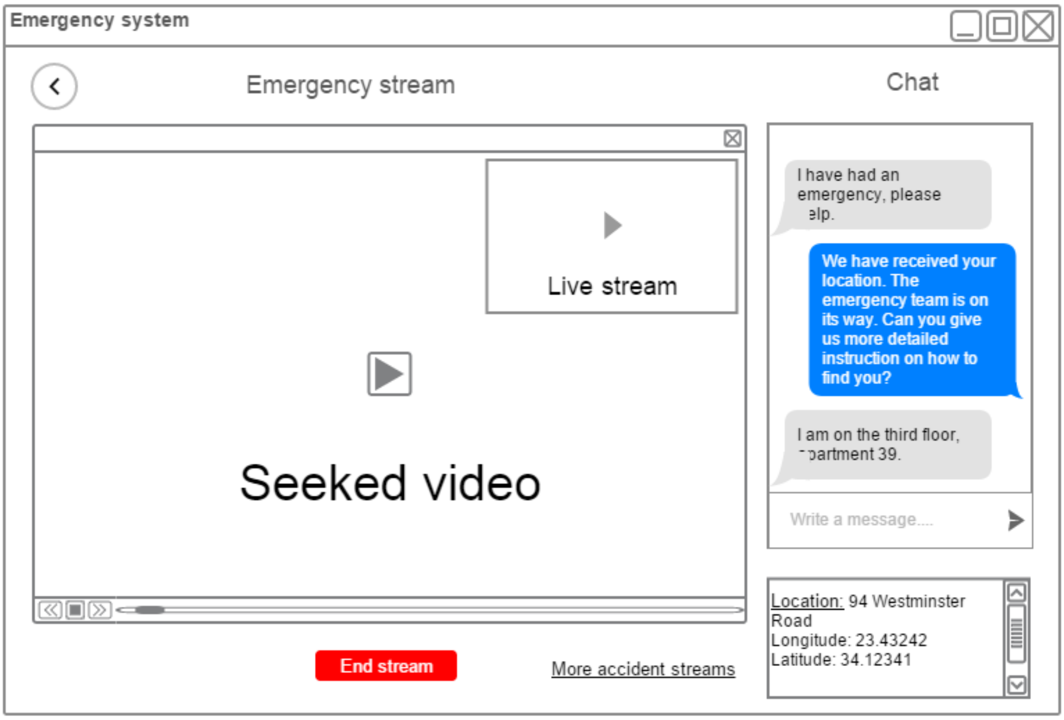
4.4.1.1 Aims During this iteration the basic components of the communication between the server and the device are produced.

4.4.1.2 Sending the location During a call, there will be several options for sending the callers location to the emergency center.

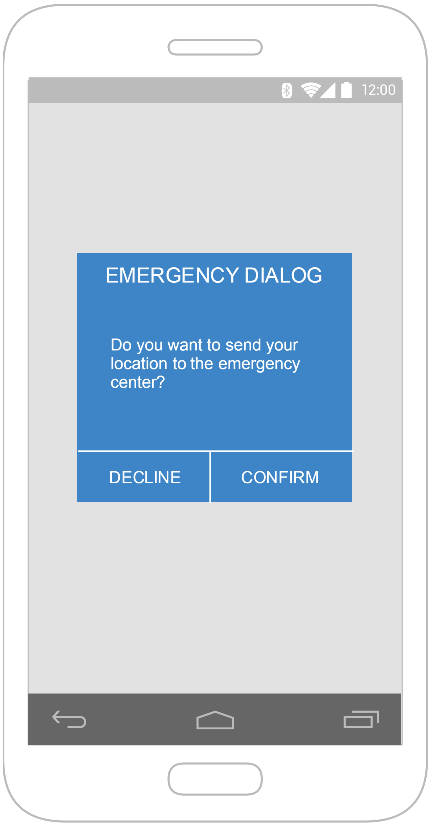
The first one will be when an operator requests the location through the backend. The user needs to have the application installed. The operator will be able to search for and start an emergency session, as he would do when initiating a video call. The UI will then provide the option for requesting the location of the users phone. A notification will then be sent over to the users phone, which will cause a popup to appear, asking for confirmation, whether or not to send the current location. After confirming, the location will be sent over to the server and the operator will be able to see the location.



Backend request location button



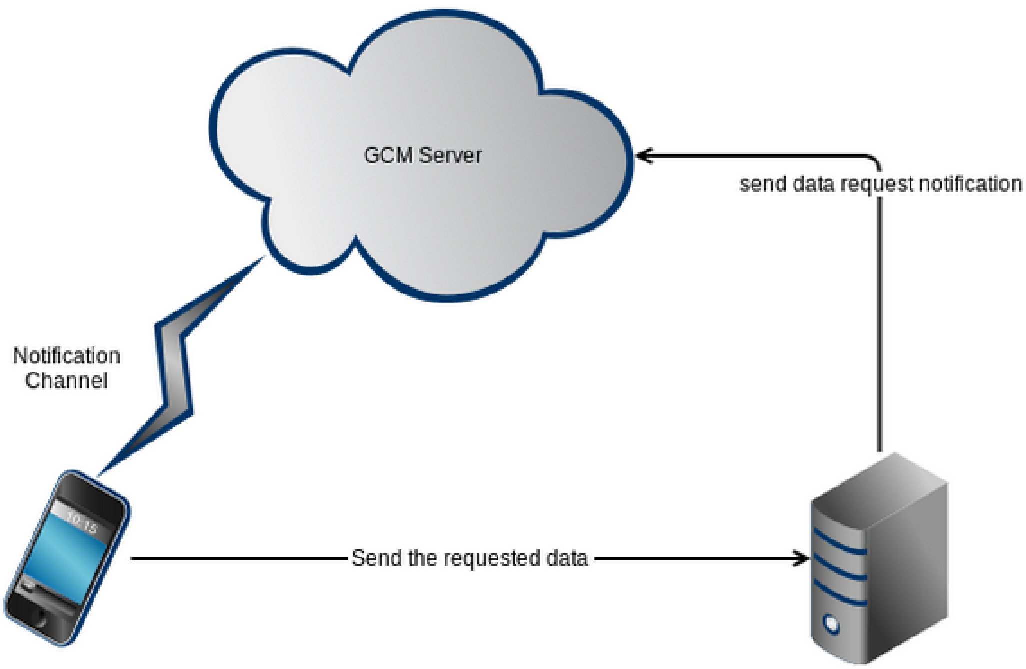
Backend location received



Confirmation dialog for sending the location

The higher level view of the communication channels of the system is shown on fig.1 . The sequence is as follows.

1. The operator/backend requests data from the mobile phone.
2. A notification is send from the backend to the Google Cloud Message (GCM) services.
3. The notification is delivered to the mobile phone via GCM.
4. The request is routed to the appropriate services on the phone.
5. The requested data is gathered/generated and send to the server.



High level communication channels

Apart from the operator requesting the location of the phone. The user will be able to do this from various places in the application: the initial screen, the UI when streaming live video or the chat UI. This will cause the phone to immediately send the location of the phone to the server and make it available to the operator.

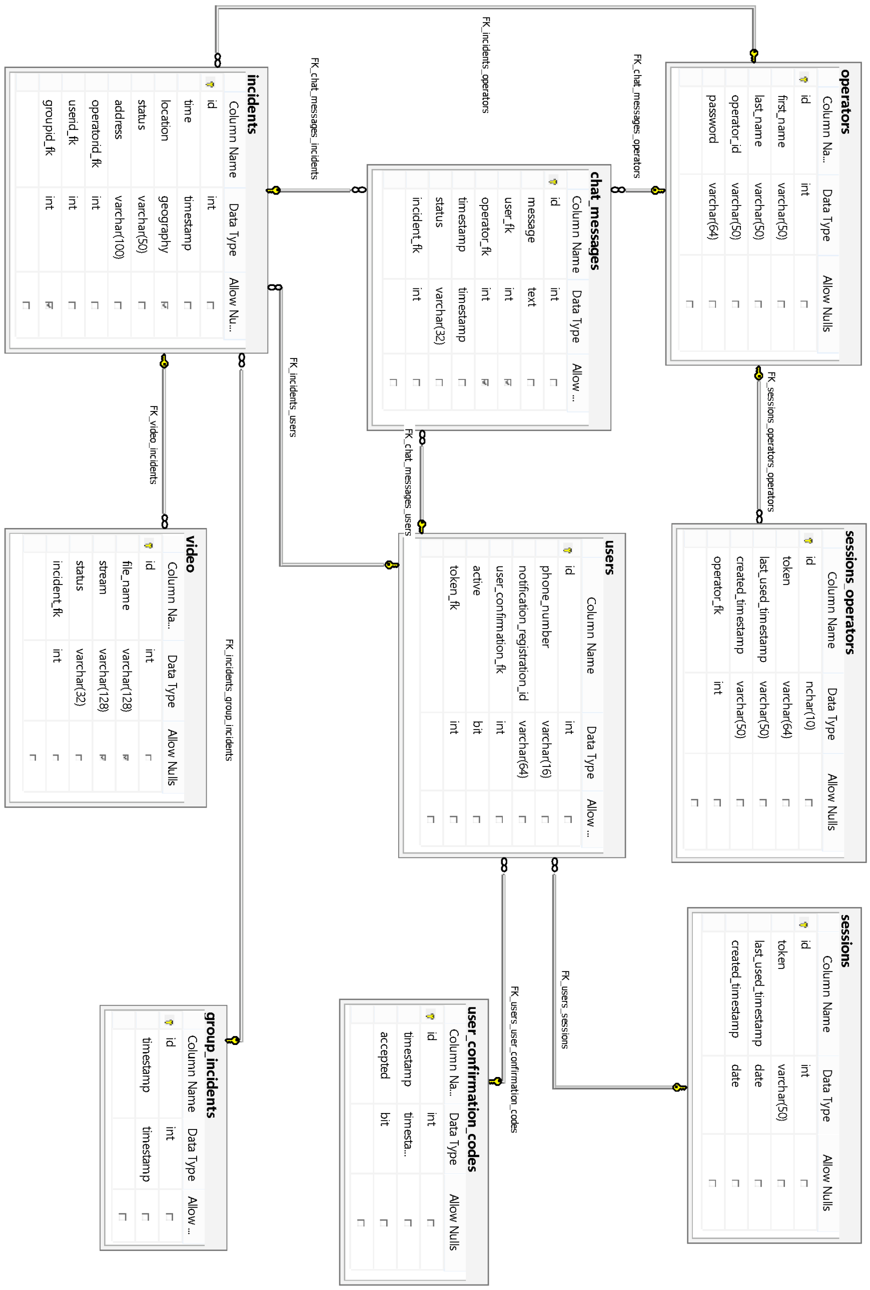
4.5 Emergency App - Database

4.5.1 Database Design

Based on all requirements from the features, that were designed so far, the following database was created. It was designed in such a way, that it would be easy to either extend it, or introduce it into currently available databases.

4.5.2 ER Diagram

ER diagram of the database.



4.6 CPR System

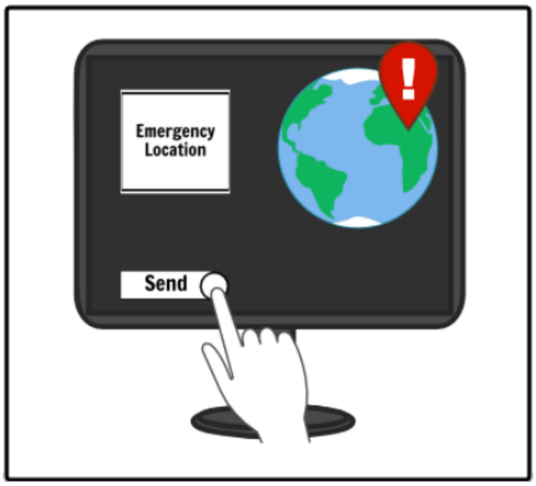
4.6.1 Iteration 1 - Basic Functionality

4.6.1.1 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.

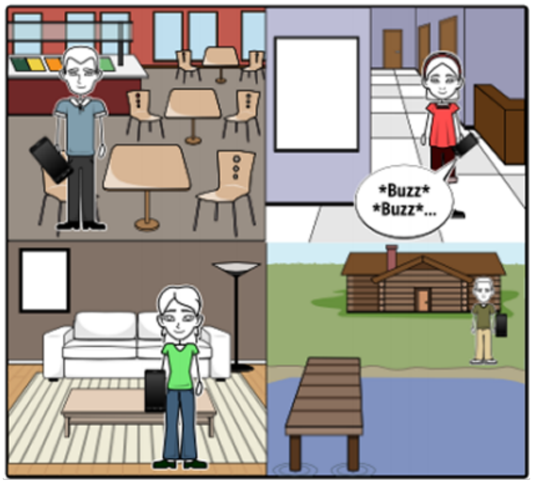
At its most basic level, the system needs to be able to send a message to a responder's devices to notify them of the location of an emergency. This individual should be the closest person in the database to the incident. They are then expected to attend the emergency location and provide assistance 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.

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

The emergency Dispatcher gets a call about an emergency situation.

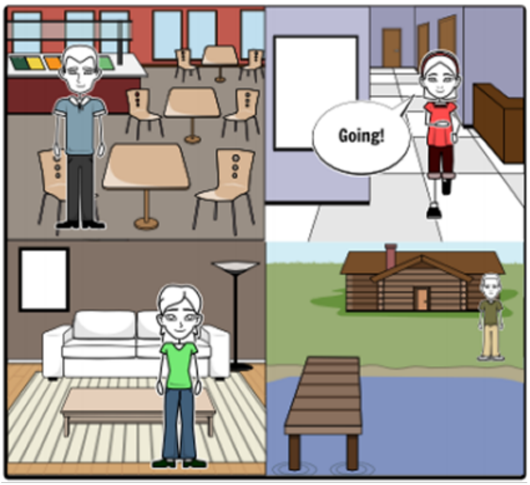


It is an appropriate emergency to ask for help from anyone nearby. The dispatcher sends the help request.

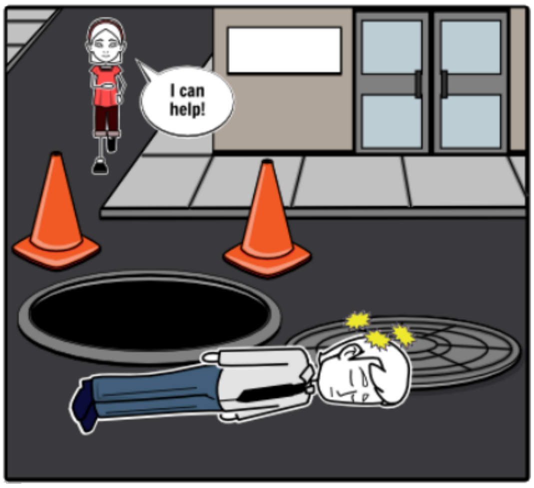


The closest person to the emergency is alerted to the situation.

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.

4.6.1.3 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"));
}
```

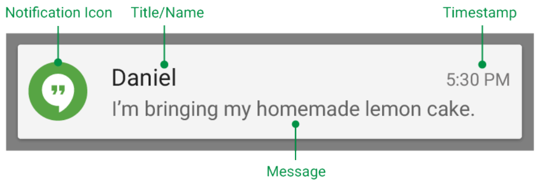
Integrating a map into our app is the route we have decided to go down as this allows us greater granularity and control over what the user sees and is able to do. This should be shown to the user when they click on an emergency notification.

4.6.1.4 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.

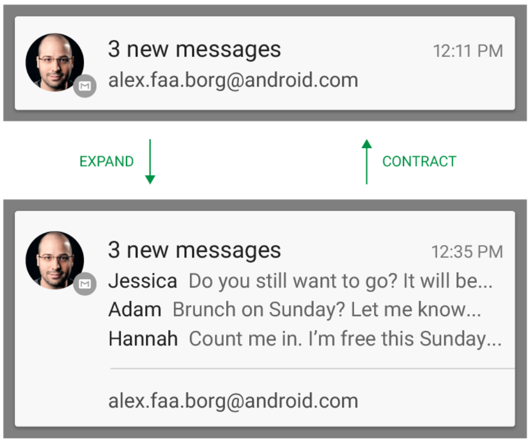
To provide both of these points to the user we will need to use Android's built-in location API, as the information provided to the device is in coordinate form. There are several features which will allow us to obtain the required information, the first is in the `Geocoder` class of the location API [19]. The method `getFromLocation()` 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 (of which we can set the longitude and latitude) 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.

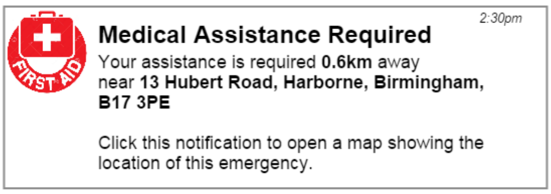


The notification icon should also be striking to grab their attention, and also needs to be unique to the app and tell the user, at a glance, the nature of the notification. Something similar to this image would be best as the image is striking and so should catch the users attention while being unique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

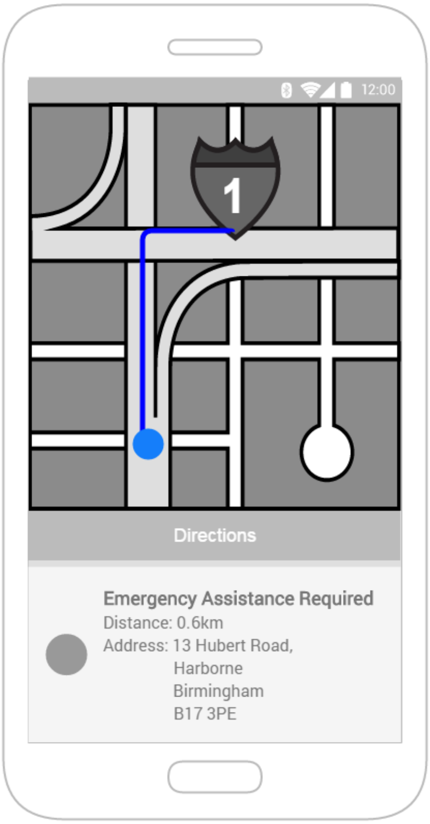
4.6.1.5 Cancel Notification The dispatchers should also be able to cancel the assistance request remotely for a variety of reasons such as the emergency has been dealt with or paramedics have already arrived to name a few. To do this we will utilise the collapsible feature of messages sent over GCM [23] 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"
     }
...
```

4.6.1.6 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 user's 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.

4.6.1.7 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

4.6.1.8 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 GCM's upstream message feature to send the message via Google's 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.

4.6.1.9 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.

4.6.1.9.1 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.

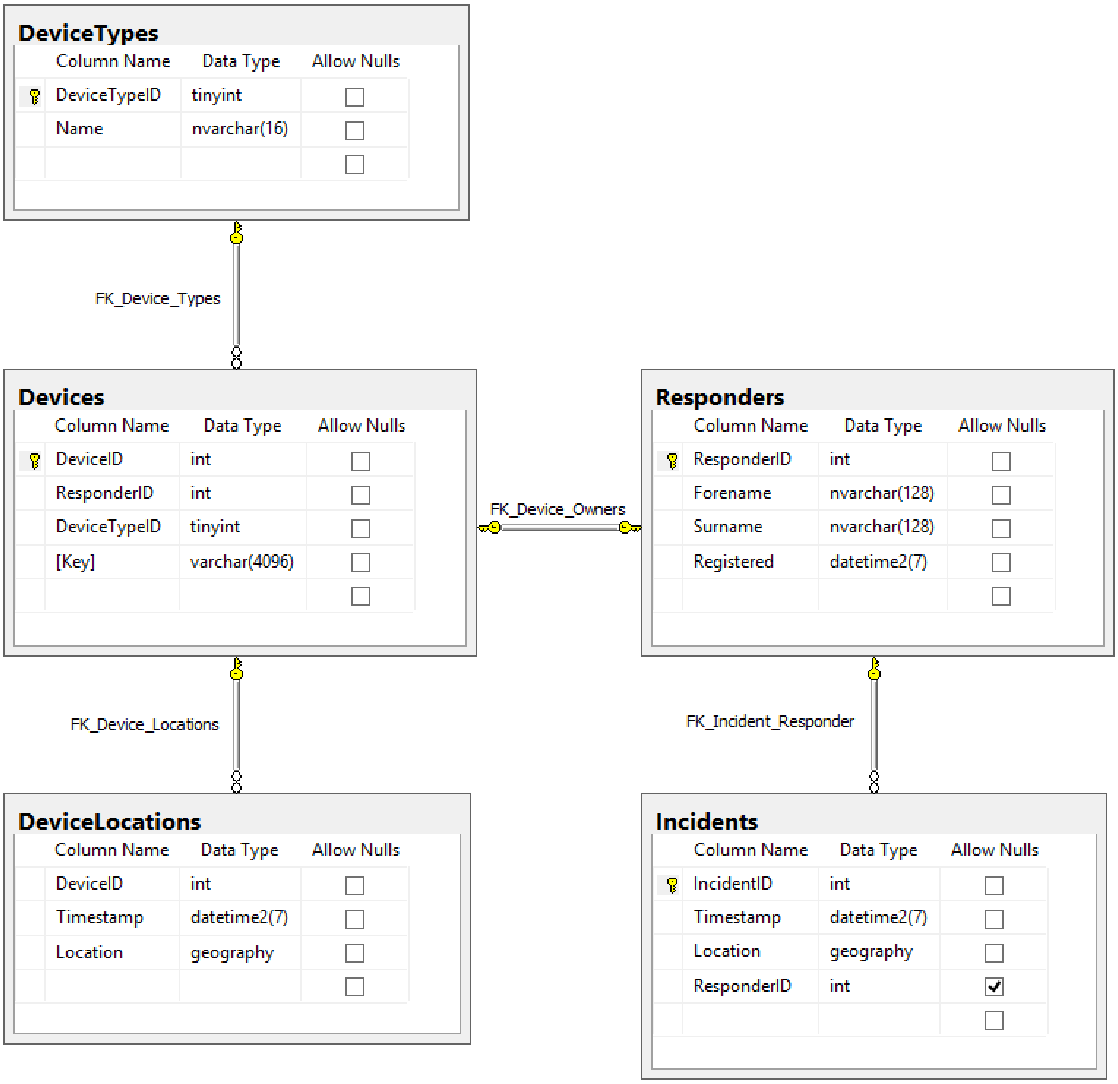
4.6.1.9.2 Data Types Location data is commonly represented as a latitude and longitude pair, however modern databases have dedicated spatial data types. SQL Server 2014 is no exception; it has the `geography` type for ellipsoidal data. The key `geography` instance type that we'll be using is `Point`: a 0-dimensional object representing a single location. We'll 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 doesn't 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.

4.6.1.9.3 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.

4.6.1.10 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;
```

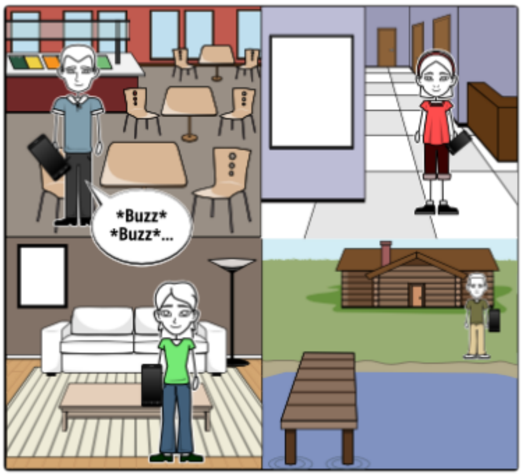
From here, it is trivial to look up the owner of the closest device and notify all of their devices of the emergency.

4.6.2 Iteration 2 - Accept & Decline an emergency

4.6.2.1 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.

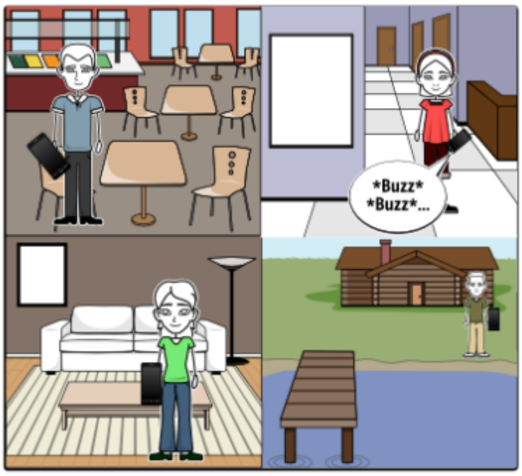
On completion, this iteration should give responders the choice to either accept or decline a request for attendance at an incident, and depending on their decision, the system should be able to react appropriately. It should also address the issue of the contacted user being unreachable after the system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

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



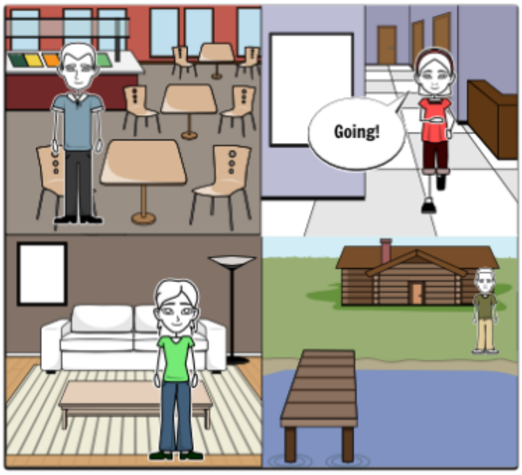
The closest person to the emergency is alerted to the situation.

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

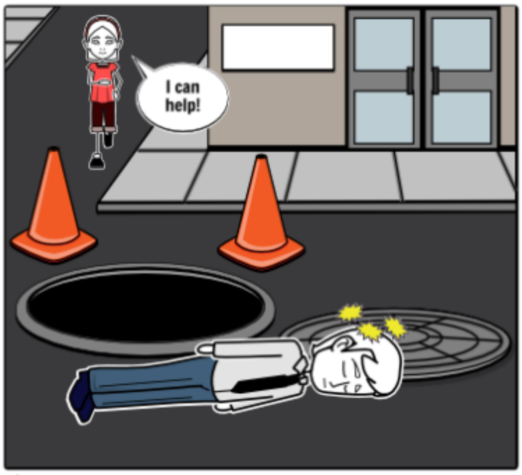


The system finds the next closest person to the emergency and notifies them.

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



The responder makes their way to the emergency.

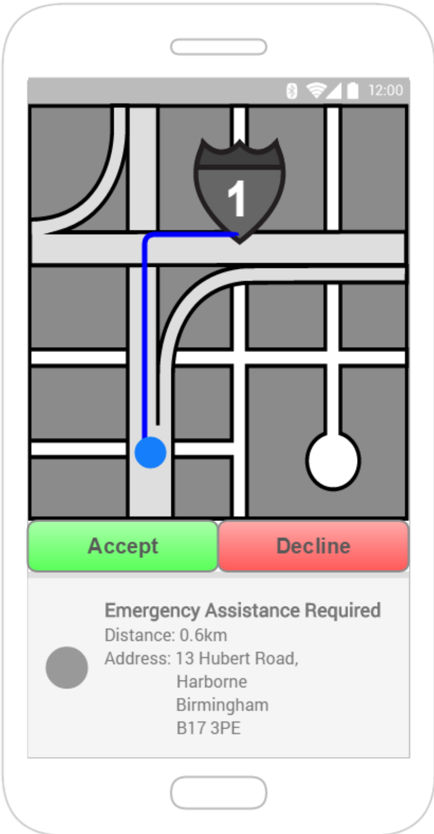


Once they arrive they help as appropriate.

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.

4.6.2.3 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 mis-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.

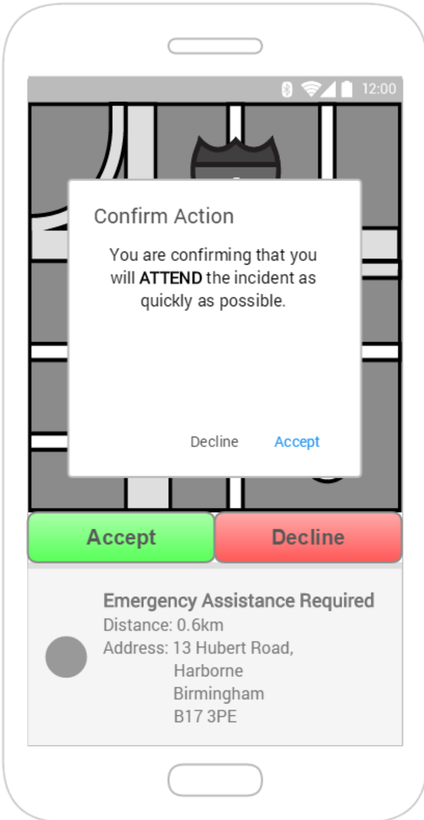
This cycles should continue until one of the following conditions has been met:

- 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.

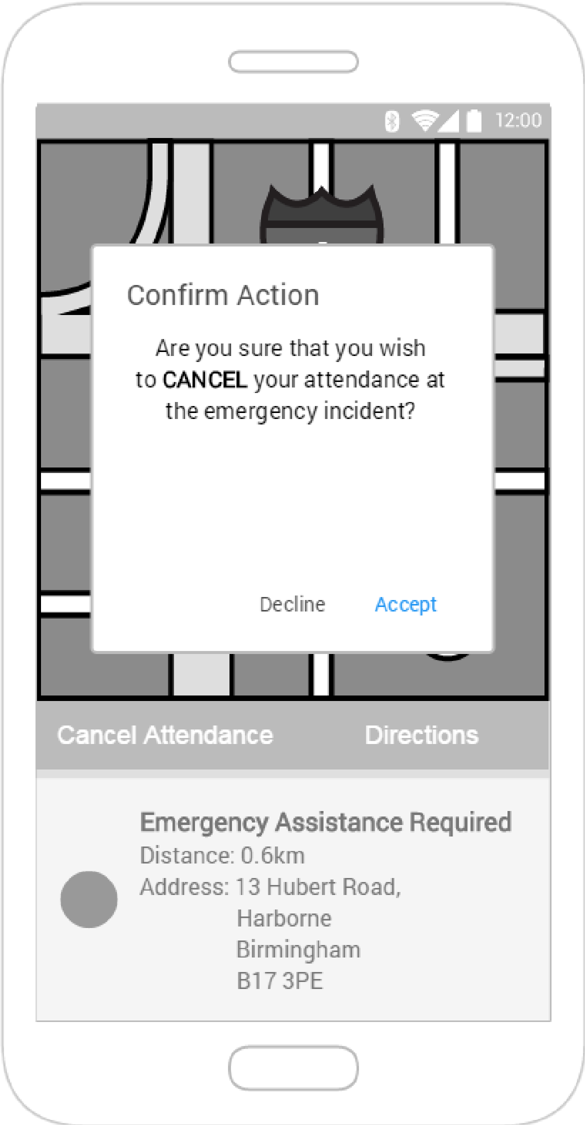
- The next individual who is due to be contacted with the emergency is beyond the reasonable response distance for a first responder. This stops the chain of help requests reaching anyone who is too far away to be able to help. This distance should be based off the time it would take an individual to 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.

4.6.2.4 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.

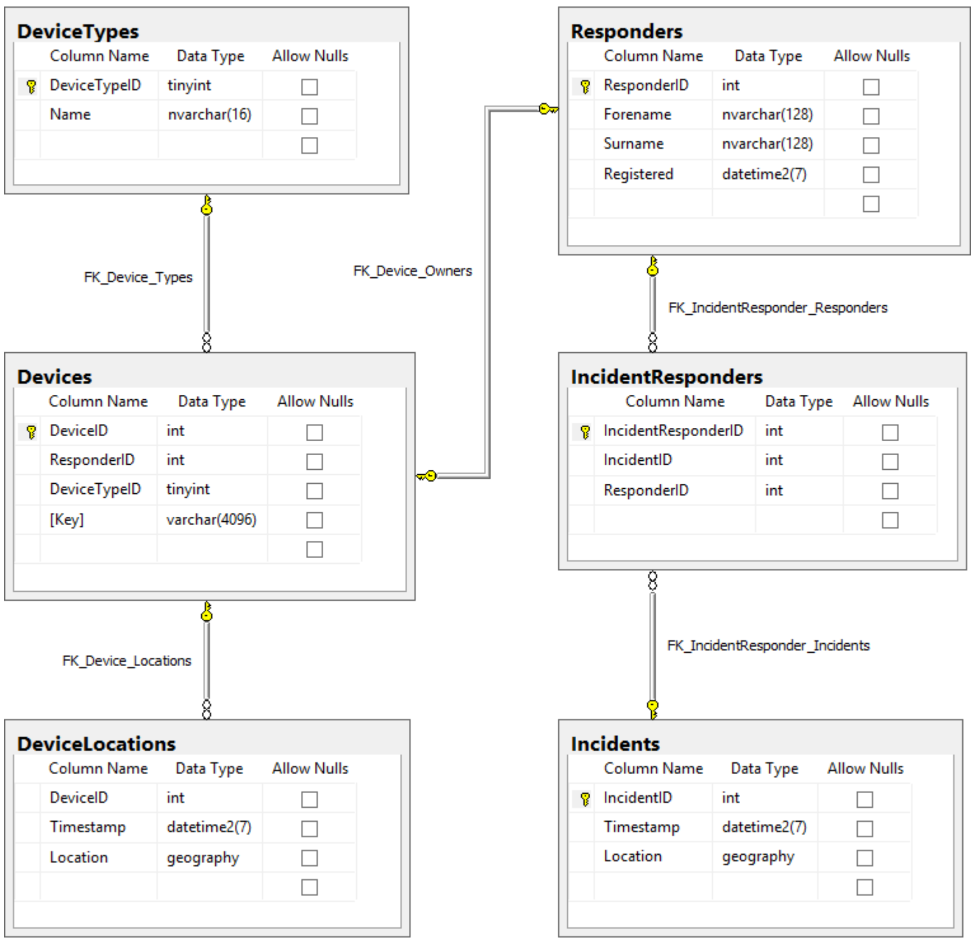
4.6.2.5 Keep playing alert sound until user accepts/declines

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.

4.6.2.6 Database Modifications

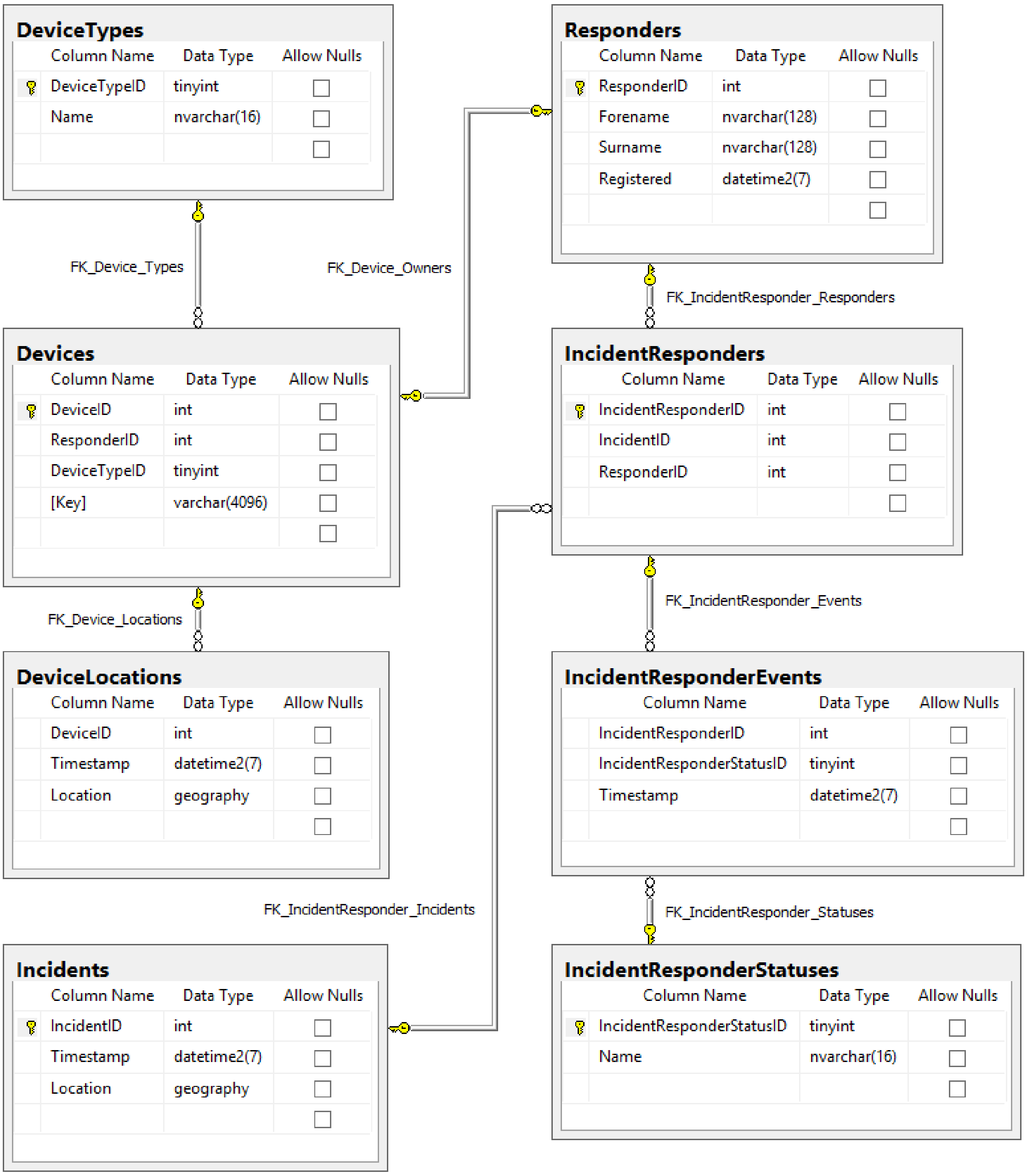
This iteration requires the ability to link multiple responders to an incident, which necessitates several changes to the database design. The first step is to remove the `ResponderID` field from the `Incidents` table, as the relationship is now too complex to be implemented in this way. Instead, another 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.

4.6.2.7 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.

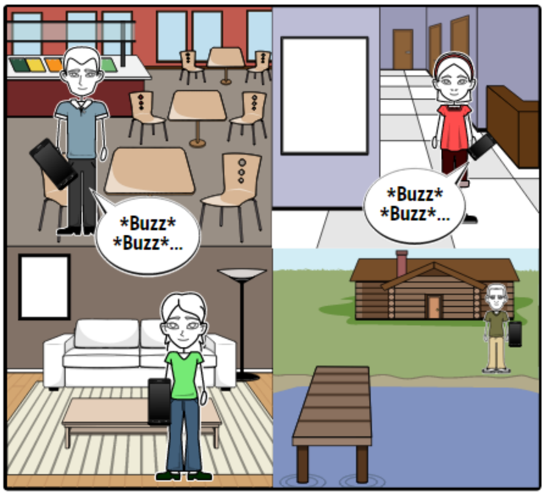
Similarly, introducing a maximum distance, above which devices (and therefore responders) are ignored is straightforward. SQL Server's `STDistance` function returns a distance in metres, so a conversion into miles can be done by dividing the returned value by 1609.344. Armed with this information, it's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

4.6.3 Iteration 3 - Improving Responder Alert

4.6.3.1 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]).

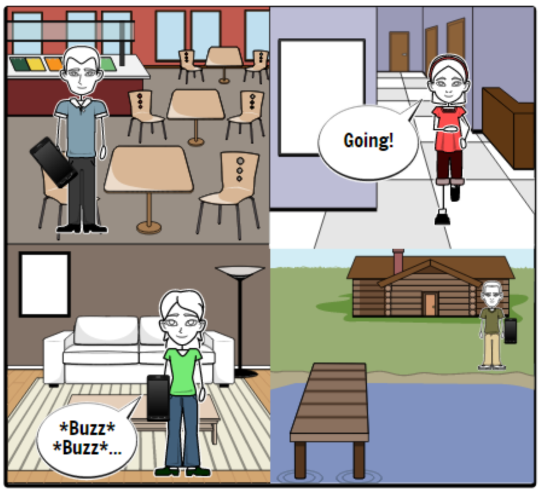
This iteration aims to reduce this concern by adding a parallel search (in the same vein as the previous iteration) so that multiple responders can be contacted simultaneously increasing the likelihood of contacting someone who is available and reducing the responder response time.

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

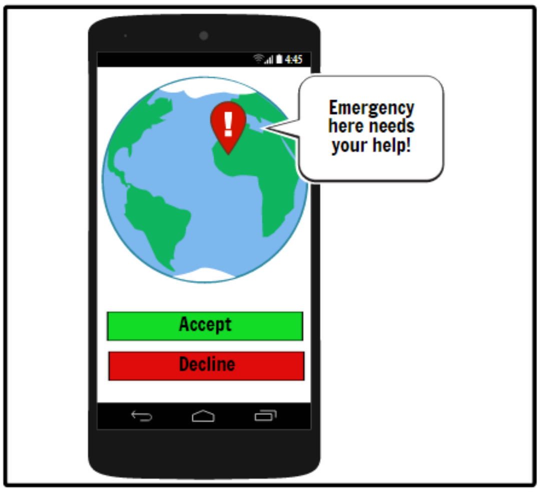


Multiple responders are simultaneously alerted to the incident.

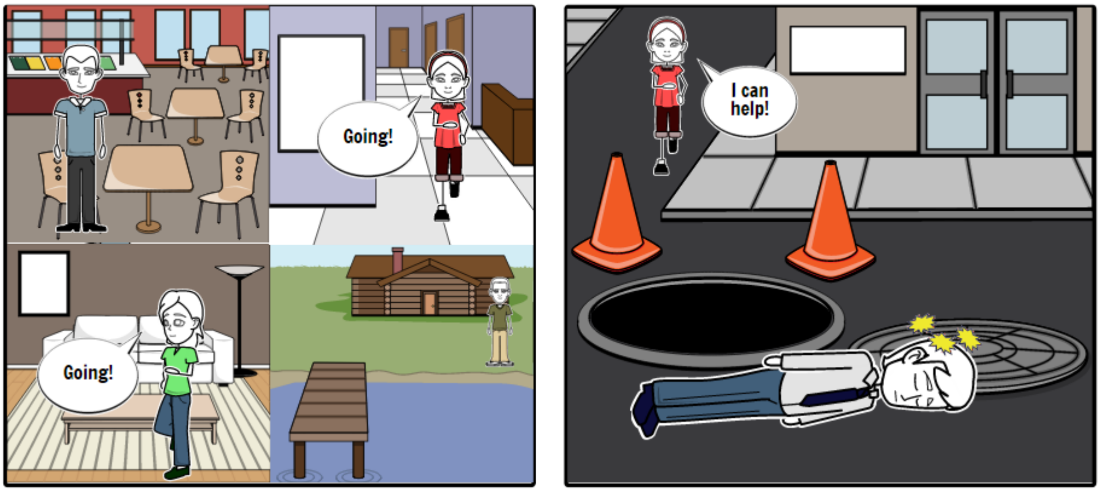
When a responder accepts that search thread is stopped.



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.

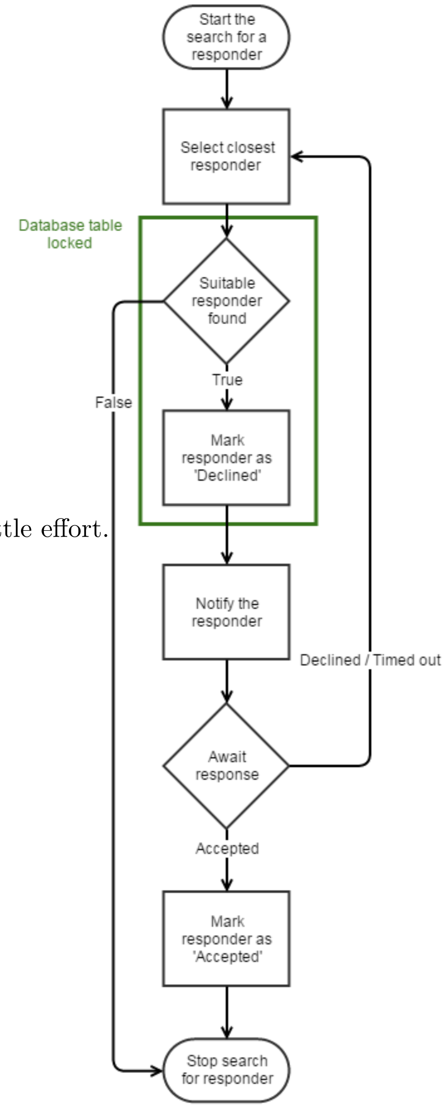


One or more responders are able to attend the incident   Once they arrive they help as appropriate.

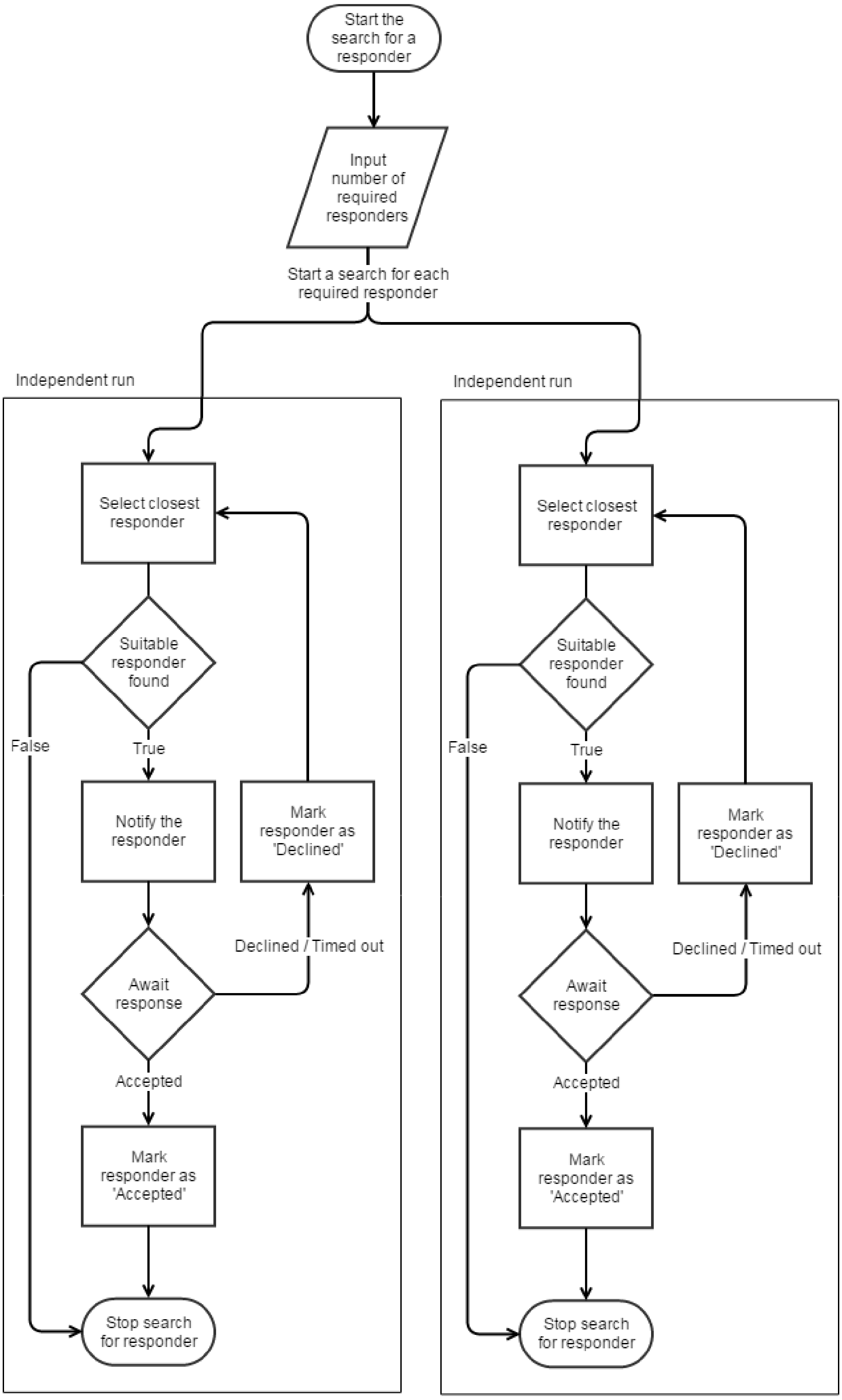
1. Multiple parallel searches for responders are started. Each stream selects from the same group of responders (without overlap).
2. The responder is given the opportunity to accept or decline the request (as in the previous iteration).
3. If they accept the request, they go off to help at the emergency. 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.

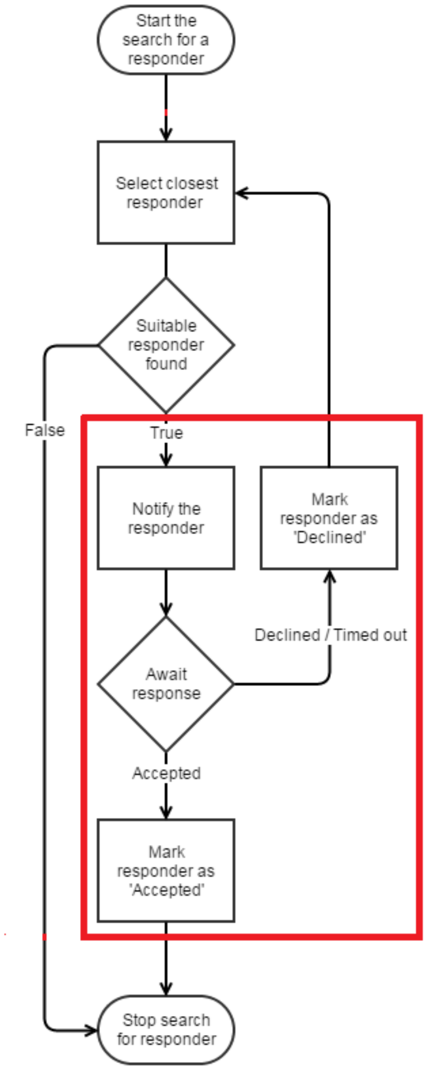
4.6.3.3 Parallel Searches In order to implement the ability to perform parallel searches for responders we need to first look at the way in which the process for searching and selecting a responder is done. Below is a flowchart showing the basic layout of searching for a responder on the server side. 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.

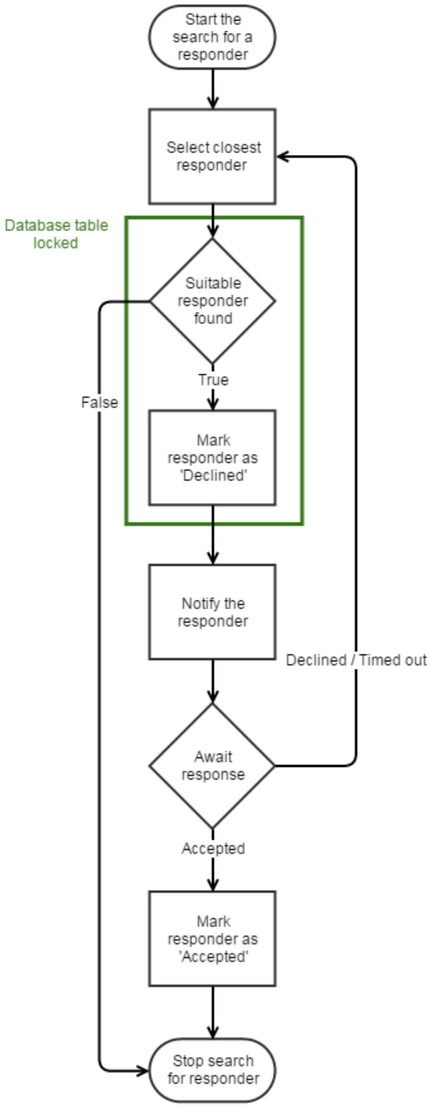


We do need to take measures to prevent some problems. The first being that we need to ensure that when parallel searches come to select a new responder to contact they do not select the same responder in the time between one search selecting the user and marking them as available or unavailable. The 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.

4.6.3.4 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.

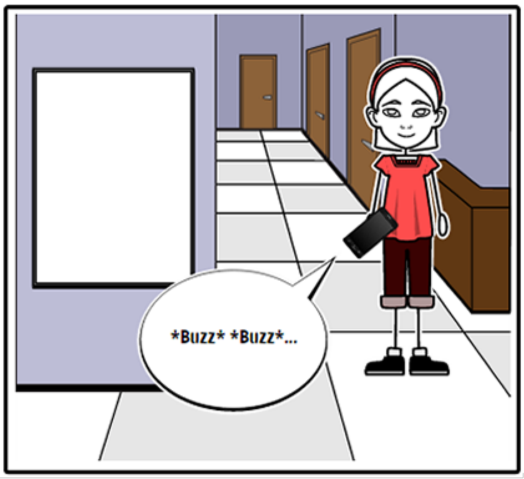
The system should also be used and, by default, contact two people for each emergency. This would mean that more potential responders are contacted faster than in the old system (through the parallelization) and it should end with two responders being sent to the incident. This is useful as it increases 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.

4.6.4 Iteration 4 - Context of the emergency

4.6.4.1 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.

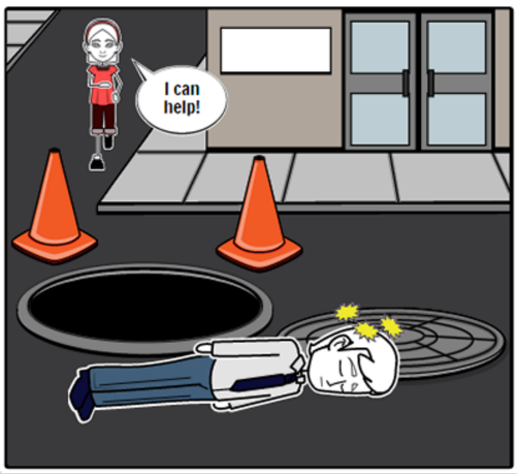
4.6.4.2 Storyboard This storyboard visualises the intended functionality of the system.



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).

4.6.4.3 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 Service's 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).

4.6.4.4 Notification Details

In order to send this new information to the first responders we can utilize the payload feature of GCM's 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"

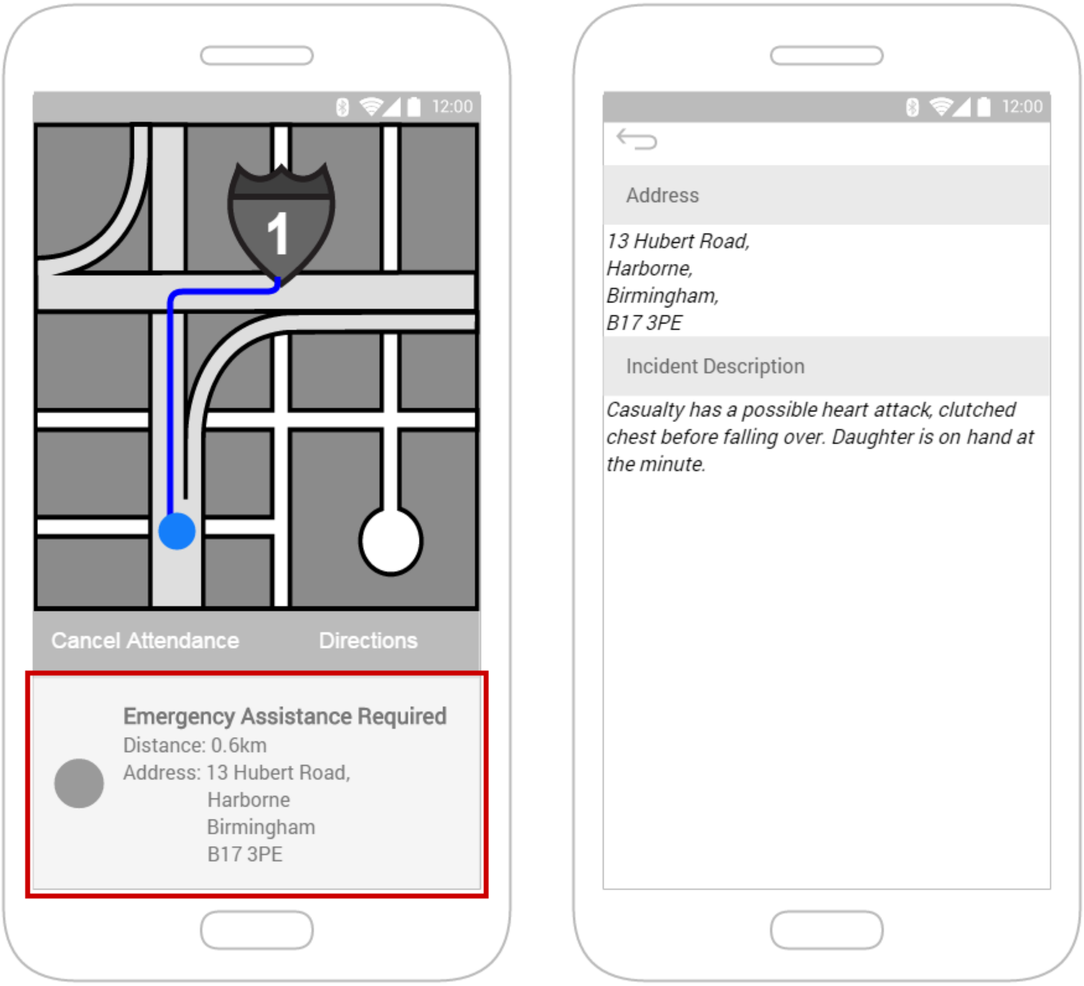
been compressed into single characters to be sent over the message, this is to help maximize the available space of the GCM payload (which is limited to 4kb) for other fields which may contain long strings. Fields with a single character will have a limited pool of available values which can be parsed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

4.6.4.5 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.

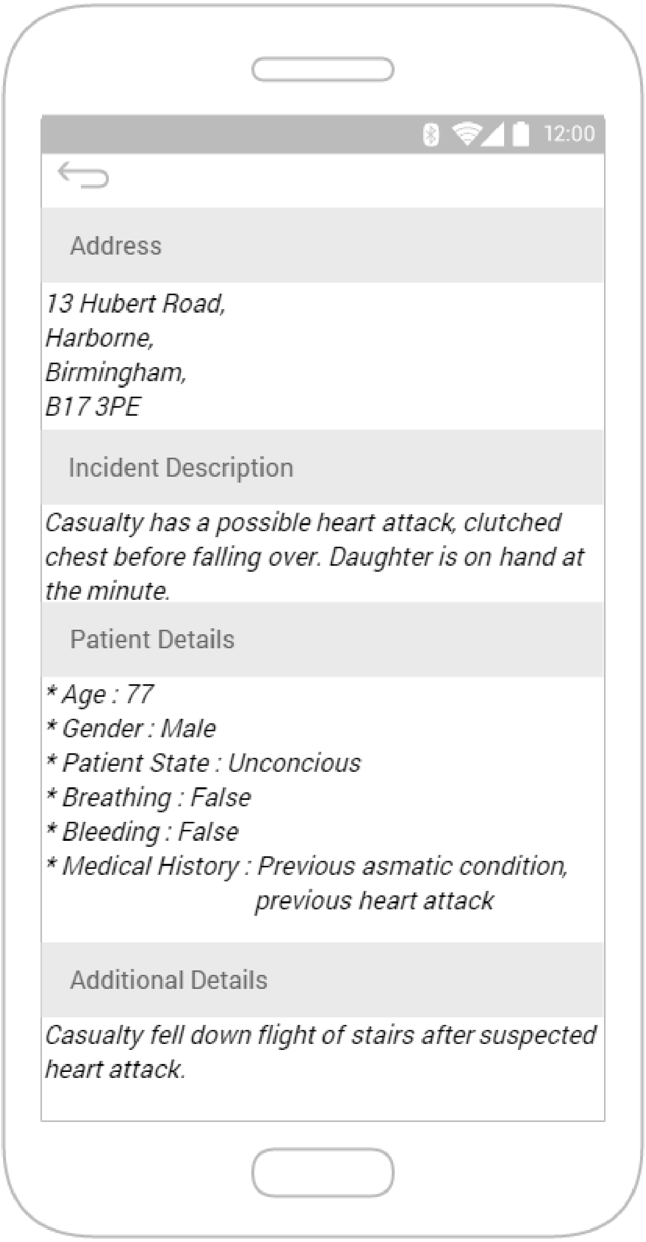


Clicking on the basic 'emergency details' will bring up the in depth details pane.

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.

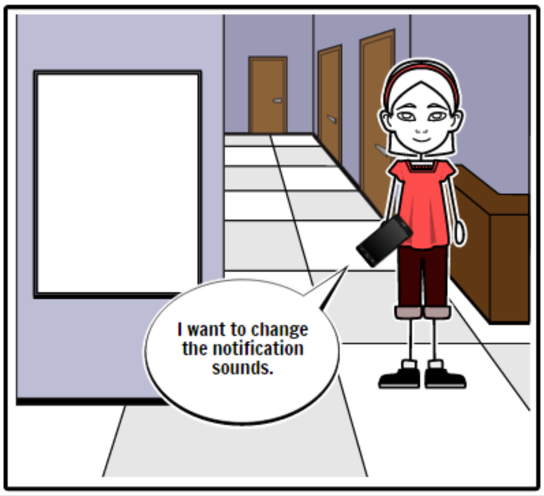


A more in depth details pane showing additional details as provided by the dispatcher.

4.6.5 Iteration 5 - User Preferences

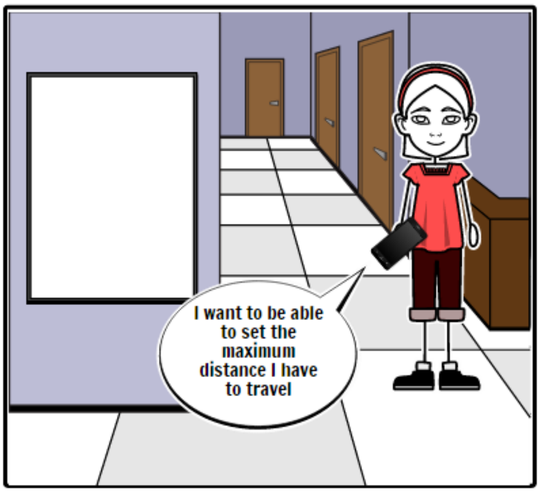
4.6.5.1 Aim This iteration gives the user more flexibility and control over various features that the CPR system offers. The user should be able to change preferences such as when they can be contacted or how they are alerted allowing them to personalise the service to fit into their lifestyle.

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

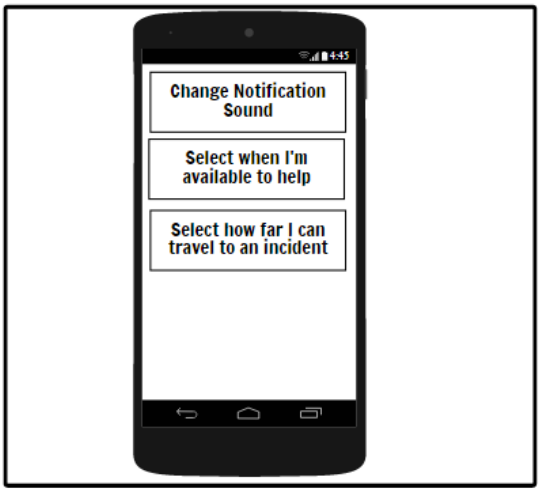


User wants to change how they can be notified of an alert.

User wants to change when they are available to help.



User wants to change how far they are able to travel to an incident.



Options are changable through a settings pane.

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.

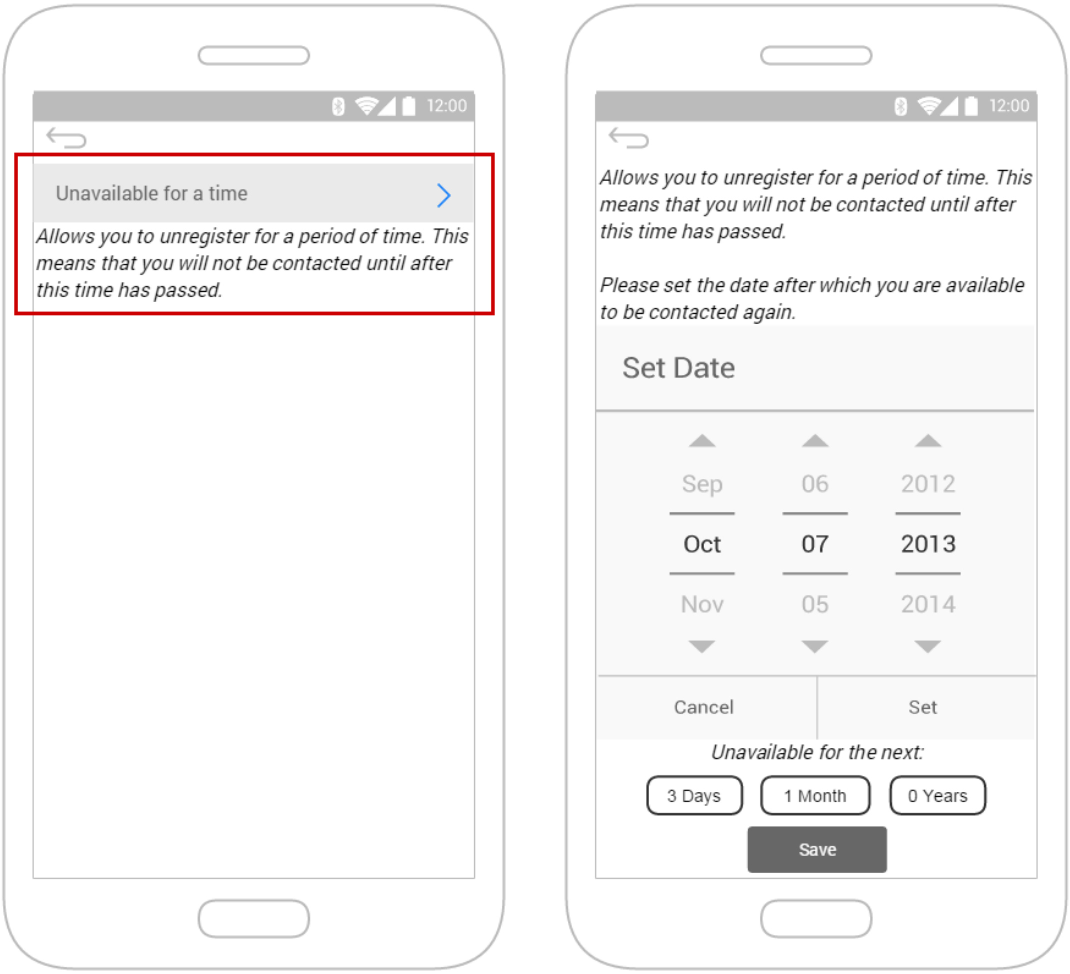
This will help the responders to emergency to personalise how they are contacted to fit into their lifestyle. There should be options to change notification alerts, change how often they can be contacted, set hours they can be contacted along with more. This means that the user has control over when 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.

4.6.5.3 '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.

4.6.5.3.1 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.

4.6.5.3.2 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.

4.6.5.3.3 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 Server's `smalldatetime` type, and the column will default to `null`.

Responders

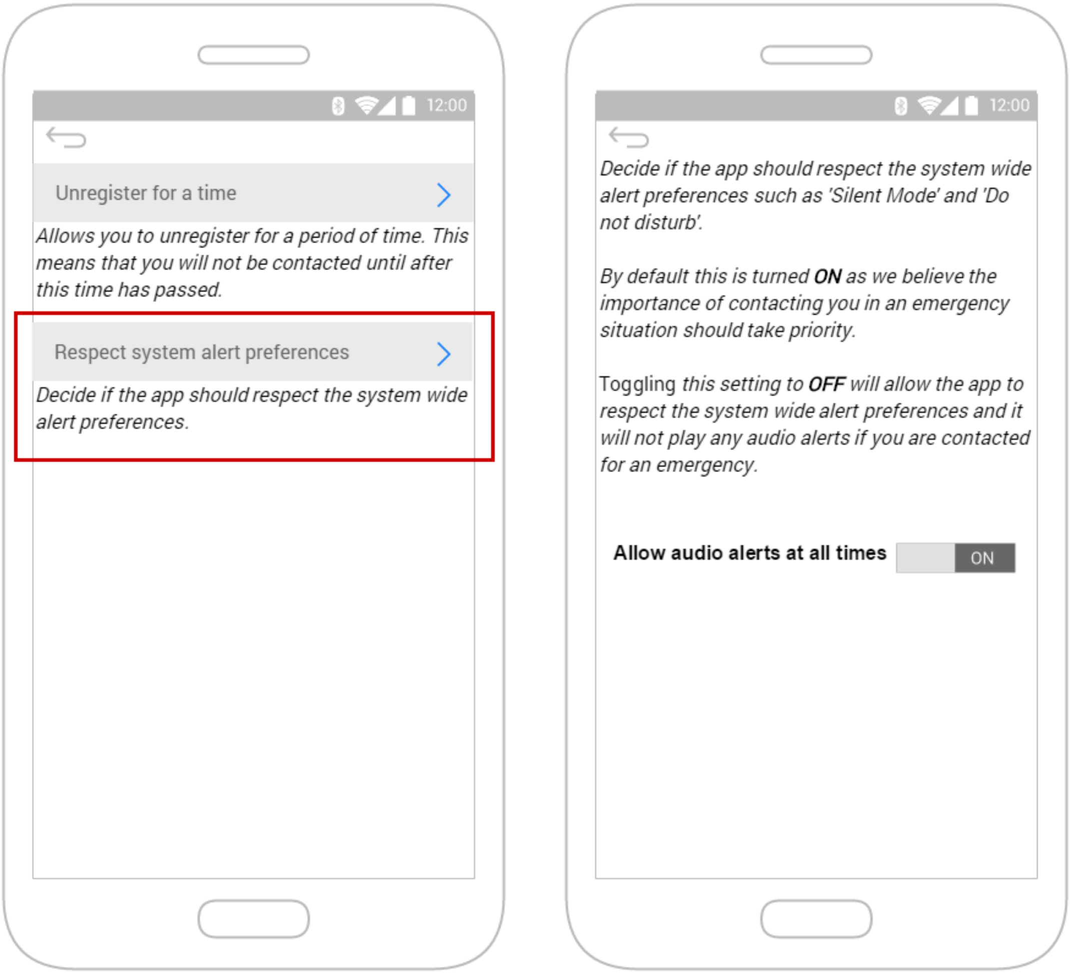
Column Name	Data Type	Allow Nulls
ResponderID	int	☐
Forename	nvarchar(128)	☐
Surname	nvarchar(128)	☐
Registered	datetime2(7)	☐
DoNotDisturbUntil	smalldatetime	☑
		☐

4.6.5.4 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 system's alert preferences. This controls whether the app should be silenced by the device's '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.

4.6.5.4.1 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.



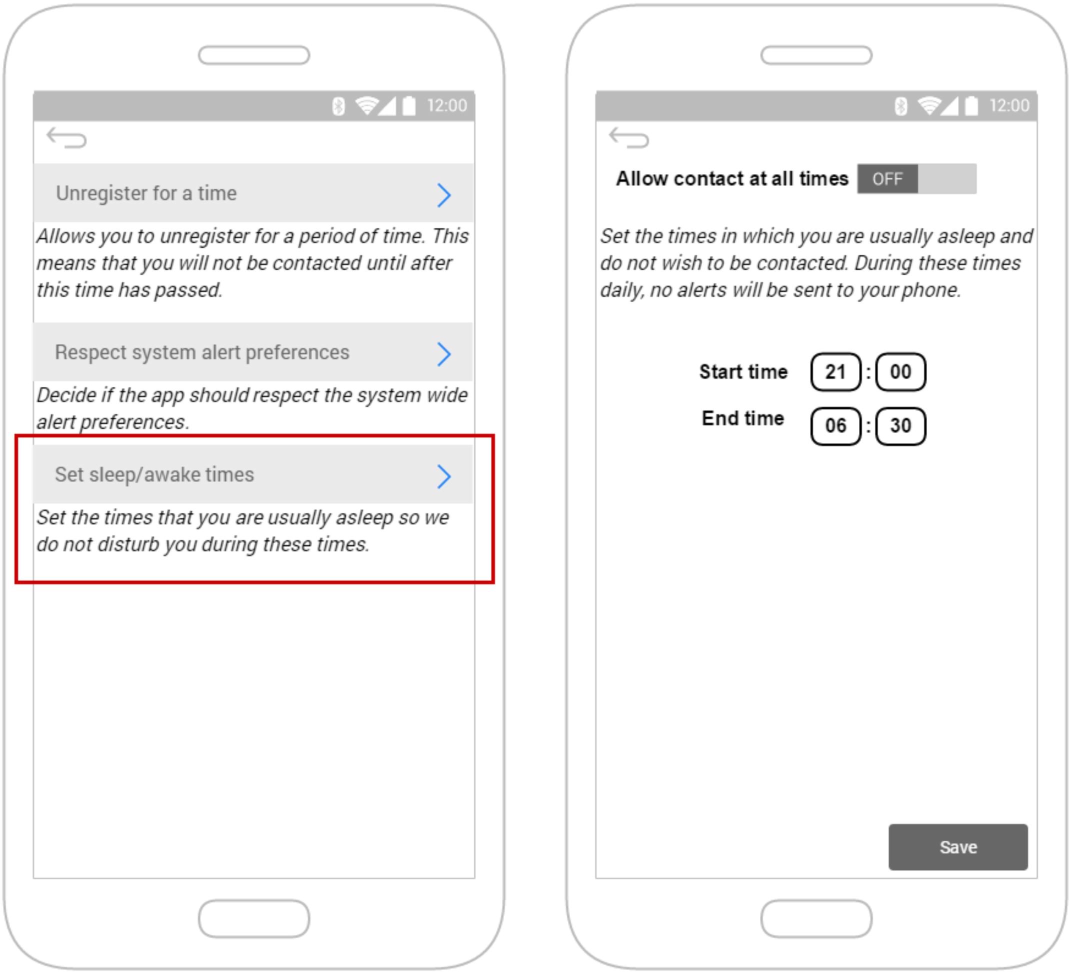
Menu describes each item below the button to change the settings for that item. **The settings pane allows the user to select if they want audio alerts at all times or not.**

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.

4.6.5.5 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.

4.6.5.5.1 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.



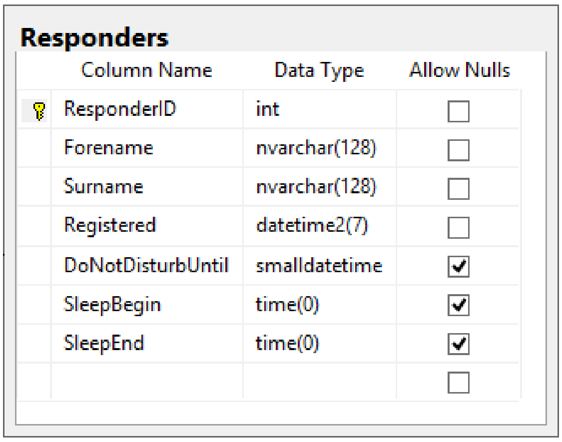
Menu describes each item below the button to change the settings for that item.

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 GCM's 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.

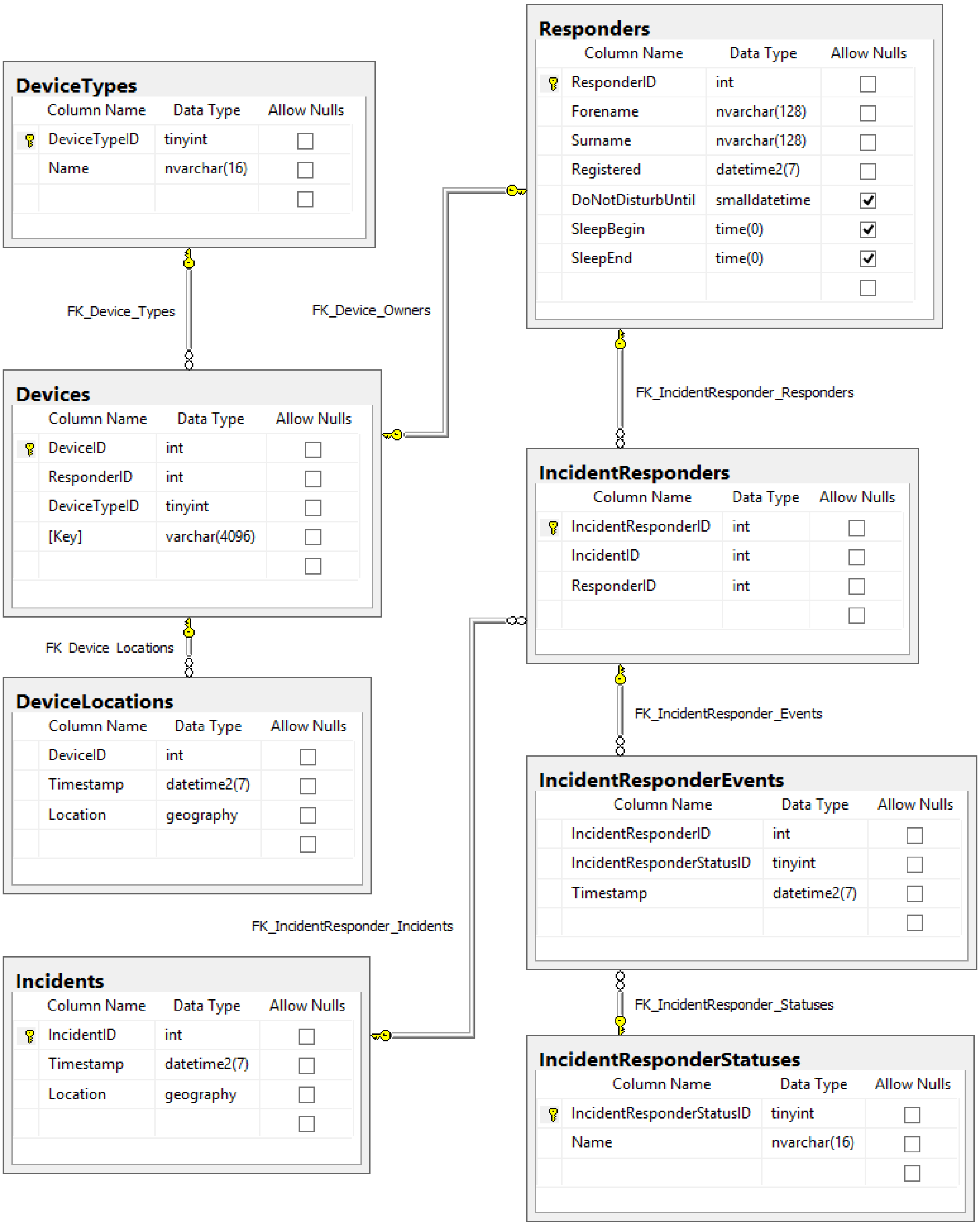
4.6.5.5.2 Server Modifications Our search algorithm will need to be modified so that once an individual is selected there is a check that the current time is not between the start and end of the 'sleep' times. This feature will provide little extra overhead to the server as there will be only one extra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

4.6.5.5.3 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 doesn't 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:



4.6.6 Iteration 6 - Improving User Selection

4.6.6.1 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.

4.6.6.2 Problems

4.6.6.2.1 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.

4.6.6.2.2 Mode of Transport Similarly the current selection mechanism doesn't 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 they're moving towards or away from an incident etc.

4.6.6.2.3 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.

4.6.6.3 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 responder's 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. People's 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.

4.7 API

4.7.1 HTTP

It was decided that an HTTP REST API will be most suitable for this project. Not only that it satisfies all the requirements but it is also extremely popular and widely used nowadays.

HTTP[57] is a text based application layer stateless protocol which uses TCP as an underlying transport protocol. HTTP request contains headers with metadata and an optional body with payload. Stateless protocol means that every request should contain all the necessary data in order to be complete successfully. No information is persisted between the requests, there is no state in the protocol. In the system being developed what is called a server side state[58] is implemented. Server side state has a lot of benefits when compared to the other session management strategies. When using this approach all of the information regarding the state is maintained on the server, the client only has a reference to that data, in our case this reference is called an authorization token and is send with every request as a header. This prevents the client from sending malicious metadata in the session. The session can always be terminated from the backend which will result in the user having to go through the authentication steps again.

The HTTP standard defines several types of requests, called methods, used for a high level distinction between the requests. In the system being developed the following HTTP methods are used:

1. GET - Does not have a body. Consists of headers only. The response contains the payload specified in the request headers. This method is used only for retrieving data. GET request should never modify any data on the server except the caching tags.
2. POST - Can include a request body. This method is used to create a resource. Typical response of a post request is the created entity.
3. PUT - This method is identical to the POST method but is used for modifying a resource.

When transferring data over a network, it has to be encoded in some way. The in-memory platform specific representation needs to be converted to a platform independent format. As it was previously explained HTTP is a text based protocol, which means that the data needs to be represented as strings. For the system being developed a JSON[59] model representation was chosen.

4.7.2 JSON

JavaScript Object Notation (JSON) is lightweight recursively defined string representation of data. Every valid JSON is either:

1. Valid JSON Object
2. Array of valid JSON Objects or Primitives

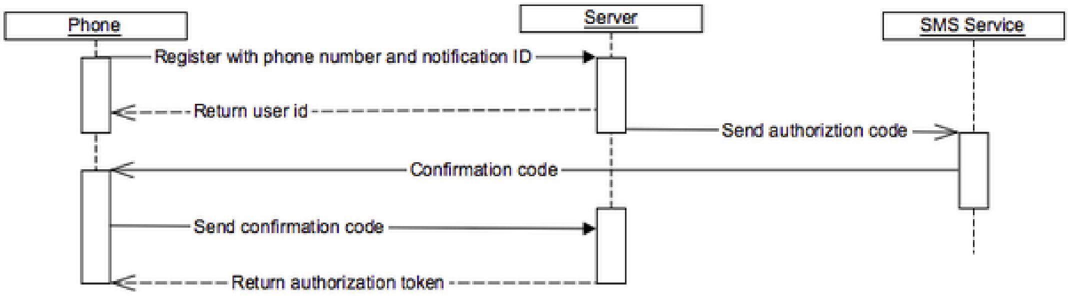
A Primitive is either a String, a Number or a Boolean and every valid JSON object consist of key of valid JSONs or primitives.

4.7.3 Documentation

As the API could be used by third party developers, a proper documentation had to be made. The Swagger Specification [60] is among the most popular specifications for REST APIs. It could auto-generate code for numerous platforms just by following the specification of the API. The swagger documentation consist of JSON/YAML specification of the HTTP Requests. Each request is uniquely identified by its HTTP Method and URL. The documentation describes the data model of the request and response objects using JSON schema[62]. All request and response models along with all the requests are defined in the appendices.

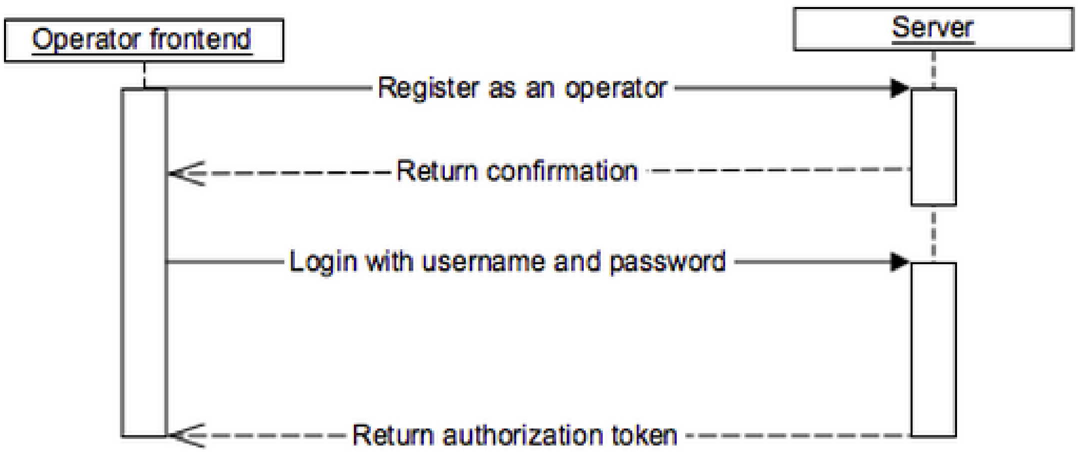
4.7.4 Security

The state of the application is maintained on the server by what is called a server side state. In order to accomplish that, an authorization process needs to be designed.



As shown on the above figure the Phone is authorized by the server using a two-stage authentication process. The mobile device sends its phone number and notification ID for GCM services to the Server. The server then creates an account and associates an authorization code with that account. The phone number and authorization code is then dispatched to the SMS Service to send an SMS to the mobile phone. Once the phone receives the SMS containing the authorization code it sends a confirmation request to the server. The server verifies the confirmation code and if valid - generates and authorization token that is returned to the device. The device uses the authorization token to sign all consecutive requests.

The two-stage authentication process is both easy to implement and secure. The usage of the SMS authorization code ensures that the device being used is real and is associated with the given telephone number. The authorization token is directly mapped to a device and in case of malicious usage of the system can be easily invalidated which will prevent the device from being able to use the system. This is a popular way of authentication when using REST APIs.



A simplified version of the operator register/login routine is shown on the above figure. As with the mobile devices the authorization is again token based. The operator registers as an user of the system providing login credentials such as username and password. Once the registration process is completed, the operator issues a login request to the server. The server then verifies the credentials and if valid - generates an unique authorization token associated with that operator and returns it back. The authorization token represents a login session that identifies an operator. Similarly to the mobile device, the operators application signs all consecutive requests with the authorization token.

The usage of an authorization token provides a way to monitor and limit the number of sessions that the current operator has. If a malicious usage is detected the authorization token can be easily invalidated which will result in preventing the access to the platform for that specific login session.

4.7.5 Requests

All described interactions with the Server in this document, if not explicitly said otherwise, are done using the API. The API contains a /incident/group request that was not described in this document. This request is intended for future use by third party systems with the ability to mark several incidents as identical.

5 Project Management

5.1 Term 1

During the first term our main objective was to research and investigate the problems with the existing emergency services. We had weekly meetings during which the tasks for the upcoming week were discussed and split between us. Everybody from the team had to research a specific topic and report back to the rest of the members. During the meetings possible issues with the existing systems were identified.

During the second part of the term we started brainstorming possible solutions to the identified problems. Each solution was broken down to very high level details. Further discussions were then conducted in order to evaluate the scope and the impact of the suggested solutions. When doing this we also involved the opinion of several people working in the emergency services as well as ordinary people that would benefit from the improved systems. Solutions that were too trivial or impossible with the currently existing technologies were discarded.

By the end of the term we managed to choose the best ideas and decided to proceed with designing them.

5.2 Term 2

Having identified the problems that we wanted to tackle we were able to start designing the solutions. Given that the requirements were not clear at the beginning of the process we decided that an Agile approach will be more suitable for designing the system. The group members were assigned to different modules of the project. High level user stories were created for all of the components. The designing process was divided into iterations where each iteration was covering several user stories. By the end of each iteration a working prototype was designed. Weekly meetings were organized where each team member was reporting the progress from the past week. Ideas and suggestions were also exchanged.

5.3 Used Services

Trello was used as a tool for organizing and keeping track of the work. Google Drive was used as a main storage and as a tool for collaboration between team members. A Facebook group was also created and used as a main communication channel.

6 Personal Reflections

6.1 George Brighton

Teamwork

After suggesting the idea for our project, I implicitly became the team leader. During the research phase, very little management was necessary, with good progress being made by all team members contacting various would-be stakeholders, and building up a detailed picture of the current state of affairs. However, the realisation of CommandPoint came as an unpleasant surprise. The effort went from building a new system to amending one being phased in, which we quickly realised was infeasible due to Lockheed Martin's unwillingness to intimate the system's finer details. As a result, we transitioned to building two applications to co-exist with CommandPoint, with minimal integration. Matt and I worked on a CPR system, with Martin and Robbie creating an emergency mobile app to aid 999 calls, and Deyan constructing an API.

Management became hard, especially during the second semester as other deadlines and projects piled up. Fortunately, effective use of Trello and Google Docs kept everyone's work in sync, while a Facebook group I created was invaluable for quick questions to avoid log jams. Work may have been slow on occasion, but we always knew where we were falling behind. Certainly, the weekly meetings and occasional Skype conversations helped immensely; Software Design Study has nailed home to me the importance of keeping track of progress and looking ahead to see what work remains to be done.

If I were given the opportunity to go back to the point where our project split into two separate applications, I would delegate responsibility for the emergency app to Martin or Robbie, and keep checking up on its progress, as well as asking for updates on the API from Deyan. I think it is fair to say that my overly hands-off approach led to a certain amount of complacency amongst everyone in the team, and caused unnecessary delays.

Activities

I found the process of investigating how the emergency services organise themselves extremely interesting, and relished the opportunity to look in detail at a real, deployed piece of software. On one hand, I was astounded by how dysfunctional the existing system was - initially we had many fantastic ideas - but it quickly became apparent that these were as hard to translate into a production system as they were innovative, especially in the case of something as critical as the technology behind the emergency services.

In particular, it was humbling to see how willing various people were to help us. I have extended particular gratitude to Carol Hunter at Northrop Grumman and Mike Brady at Lockheed Martin, who were instrumental in gathering information and opinions on CAD and CommandPoint. Their contributions made us aware of the inevitability of CommandPoint, and helped steer us towards our final choices of application.

Summary

This module has been without a doubt the most eye-opening in terms of commercial use of software. Very distinct from any other I have taken this year, it has provided me with a much better idea of how systems comes to exist, and the challenges that must be overcome before even a single line of code is written. In the same way software engineering investigates how systems are implemented, I see Software Design Study as an equivalent for how system ideas come to exist in the first place.

Despite several hiccups, we worked surprisingly consistently and everyone made a valuable contribution to the project. Some things were done very well, while others have been a fantastic learning experience. Overall, I would happily take the module again.

6.2 Matthew Flint

Our first challenge was picking a topic that could sustain us throughout the entire year. We came up with a number of good ideas but ultimately decided to settle on a system for the Emergency Services as we had some vague knowledge that the current one was not very good and we thought it provided enough scope to really be able to explore.

I initially thought that with such a large system currently in place and it being used in a very publicly central service that it would prove relatively easy to research what the current system did, how it worked and any problems associated with it. This was not the case, obtaining information of this kind proved extremely difficult and very little of it is accessible via the internet (making the task even harder). Eventually we did find that the current system was so bad that there was already another (command point) being developed to take its place.

This placed us in a dilemma, from which we took a long time to decide how to proceed. We were unsure whether we wanted to build a system better than the current one, though as command point was already in development this seemed rather pointless. Then we looked at improving on command point, however it seems that in the most part they designed and developed the system well and left us little scope to do this. Eventually we decided that we should improve upon the system by thinking of new and innovative features that could benefit people using the system on both ends of an emergency. This proved to be a good direction to proceed in as it gave us a large scope to think of new ideas to help the current system and allowed us to be more creative in our design.

We now had an idea of what to do but were not very sure about how to start the project. We wasted a long time deliberating with no real organization until we finally decided to employ an agile scrum approach and delegate certain tasks to different people. Once this was done and we had a clearer idea of the direction we were heading in things started to move along but very slowly. This project took a long while to get going but once it did finding the next task to do was easy.

The amount of time and effort put into producing this document was considerable, this is the project that I have been a part of and easily ate up hours on weekends especially towards the end of the project.

Overall I feel that we have produced a good report that a software development team could take and produce a system from. The project was not without its challenges but we have overcome them as a team and made it through.

6.3 Deyan Genovski

When the module was initially introduced to me I was confused as to what type of system we had to design. After having some initial meetings with my teammates we decided that a community project would be both interesting for us and suitable for the type of documents we had to produce. We had a couple of ideas but eventually we chose to investigate and improve the way that the emergency services currently work.

At the beginning of the first term our task was to research and investigate how the current emergency services work. During this research process we had to identify possible errors or flaws with the current way of managing emergencies.

As it turned out, a lot of the information regarding the emergency services available on the internet was either outdated, inaccurate or extremely vague. This in terms meant that we had to get in touch with some of the people that use or develop the existing infrastructure. This was challenging for us, because as we realized some of the people we were trying to contact were extremely busy.

While researching the way the emergencies are managed we found that there were a lot of flaws. There is however a system called Command Point that is actively being developed at the moment that would solve many, if not all, of the issues with the current system.

At that point we realized that it would be rather unnecessary to try and fix what is already fixed by CommandPoint. That is when we had to think in way that the developers of the current systems have not. We decided to use the technologies provided by modern mobile phones in order to help dealing with emergencies.

This whole process was challenging for me but taught me that I always have to try and look at a problem from a different perspective. Even though that some perspective may seem ridiculous at first, it is surprising how often it turns out to be beneficial.

During the second term we had to design the systems that we chose. There were a lot of challenges both when organizing ourselves and when designing the actual product.

While designing our product I improved my understanding of the technologies that we had to use as I had to carefully read the documentation and produce written justification as to why we have chosen a specific technology. I felt that at times it was more difficult and time consuming to write a justification for using certain technology than to actually implement it.

Even though that I am person that prefers developing rather than planning, this module helped me understand that designing a software system is a difficult but important process.

Looking back at the project, there are some things that I would do differently but I am quite happy with what we have produced.

6.4 Martin Mihov

At the beginning of the module, I had a really vague idea about what was intended to happen during the year. While choosing a topic, we all came up with good ideas and we all wanted to focus on a community project, rather than a commercial one. This helped us identify and agree on a topic quickly. After doing a bit of research, we found out that the current way that the handling of the emergency calls works is outdated and has a lot of issues.

Having decided on a topic, we did further research and we managed to identify even more and more issues with the emergency services systems. By the middle of the first term, I was a bit more confident that what we were doing was going to be interesting, as well as challenging. It turned out to be both, but rather more challenging than I first thought. I found out that researching a topic was not as easy as I imagined. And this was especially the case with the emergency services, as most of the information online was outdated. This caused us to contact people working at the emergency services in order to actually get some feedback on the current systems.

Having identified many issues, we were ready to start thinking of solutions to the problems. The various technologies that I have learned during my time in university and I faced during my work experience, helped me to identify many solutions. Some of them turned out to be not very efficient, however, others were really innovative. By the end of the first term, we managed to focus on specific issues and choose several solutions that we were going to include in our software design, with which I was really happy.

During the second term we started actually designing our solutions and as we were not sure what we were going to end up with, we decided to go for an iterative approach. During my time in university, I learned about several software engineering approaches, however, iterative approaches were not famous with producing quality documentation. This made me do research on what was the best way to keep our approach iterative, while keeping a good log, which I could later use in the final report.

Apart from that, while designing the technical details of the solutions we chose, I did a lot of research of available technologies which were required for a specific part of the solution. This improved my general knowledge about how various things work and what are some of the state-of-the-art technologies out there. Documenting and evaluating the choices that we made through the second term, improved my confidence with software design.

Working in a team was another challenging task that I faced during the module. Being in a team of 5 people meant that we had to organize ourselves and split the tasks, so that each of us was assigned an equal amount of work. We tried to use everyones best skills in order to do what was best for the project. Having regular meetings and ensuring that everyone was happy with what was going on and the decisions we made, was very important, in order to work efficiently. In some cases this was very difficult, as some of the tasks were totally unrelated and sometimes I felt like we were working on different projects. However, in my opinion, everyone from my team did very well, showing impressive knowledge and skills in some part of the module and showing weaknesses in others.

Looking back at the project, I see a lot of things that could have been done better, both in terms of organization and in terms of software design approach. However, identifying those mistakes, make me feel confident about the skills that I have gained throughout the module. With this experience I will be able to be more efficient in my future projects. Apart from that I will be able to identify possible problems or conflicts earlier during a project, which will give the opportunity to deal with them easier and with less impact to the project.

6.5 Robert Zlatarski

At the beginning of the module, we were given a task to choose a project, on which we should later on produce a software design. We had some good ideas, but we all agreed that we should focus on something that could improve peoples lives. Then we came across on how emergency situations are handled nowadays and saw a lot of weaknesses and the use of outdated technologies. We decided that we should create a product, which would ease up and improve how emergencies are handled. There was still the question, should we improve the current system or create one of our own. On mutual agreement, a new system was chosen to be built, which had to be easily extendable.

Doing a lot of research, conducting different surveys and speaking with professionals, we faced different problems and factors that could impact the functionalities of our system. I did not have any idea of how emergencies were handled in the UK and I learned a lot of new interesting stuff on the current systems. We all split the work equally and had regular meetings two or three times a week to discuss our progress and combine our ideas. Eventually, by the end of the first term, we had a clear vision on what we were going to build, having accepted a lot of different innovative ideas and refused others.

At the beginning of the second term, we had to start writing up the documentation. We faced problems with choosing the software design methodology, as we had to pick either an iterative agile approach or waterfall model. Eventually, having put a decent amount of research, we decided that we should use an iterative approach. Then we started working and researching the technologies that we were going to use. It was not easy to pick the right technologies for our product. Having gone through a lot of studies and tests, we had pick those, which would suit our project the best.

We all put a decent amount of work on the document that we produced, facing different problems and learning interesting new stuff. Sometimes, organizing ourselves was not easy, as we had other university duties and some people were busier at a certain time than others. Nevertheless, we were spending quite a lot of time working on it, especially towards the end of the term. Working in a team of five people was sometimes a bit challenging to me. Sometimes our documentation was a bit inconsistent and those were the main things, which we discussed and did in the meetings we had. Having eventually handled that, we were making a good progress. By the end of the Easter break, most of our documents were ready and had to be structured together.

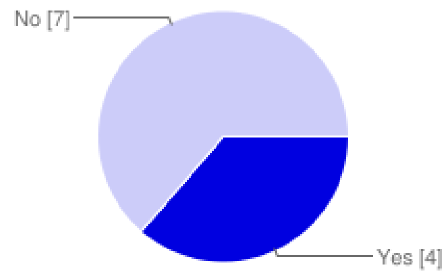
Overall, I am happy with the product that we have created. It turned out to be a system that could really save peoples lives and improve the current way of how emergency situations are handled. I have gained a lot of useful experience and improved both my team working and software designing skills. Moreover I am aware of technologies, which before I havent even heard of.

7 Appendices

7.1 Emergency App Features Survey

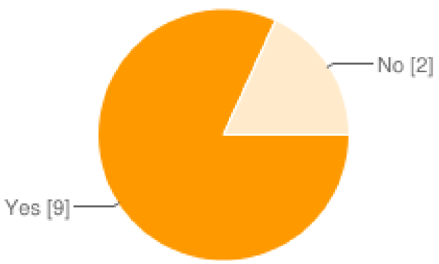
Pages from the emergency app survey.

Have you ever called 999 or 112 to request help in a case of emergency?



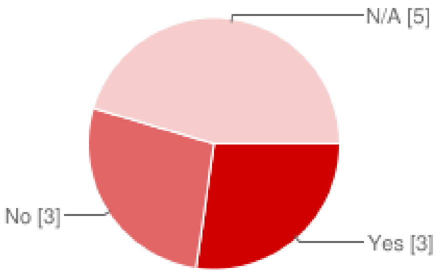
Yes	**4**	36.4%
No	**7**	63.6%

Did you use your mobile phone, or if you have not called, would you be using it for the manner?



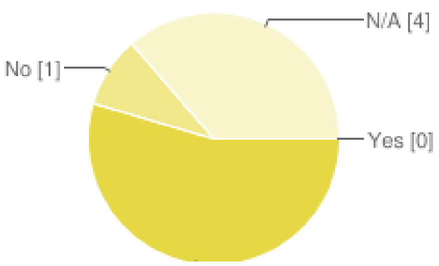
Yes	**9**	81.8%
No	**2**	18.2%

Were you able to tell exactly the location where the accident happened?



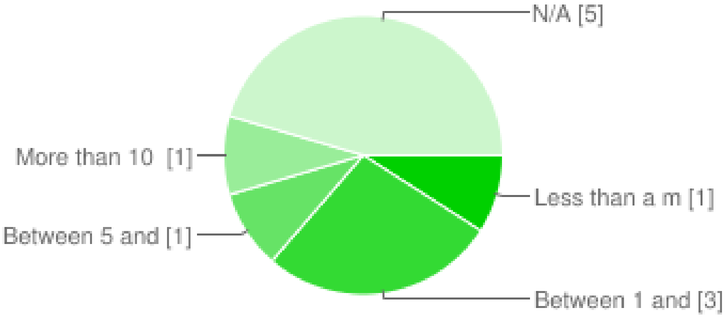
Yes	**3**	27.3%
No	**3**	27.3%
N/A	**5**	45.5%

Did you find it difficult to explain what has happened?



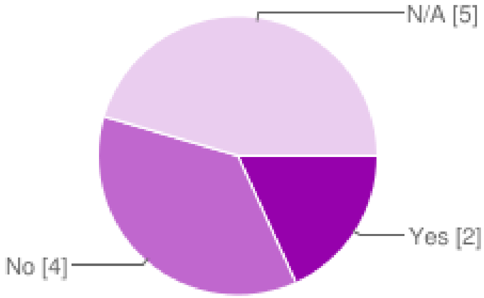
Yes	**0**	0%
Slightly	**6**	54.5%
No	**1**	9.1%
N/A	**4**	36.4%

How long was the call?



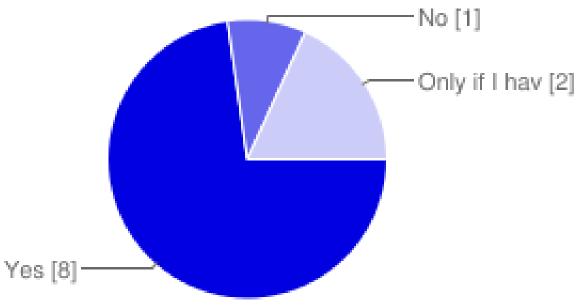
Less than a minute	**1**	9.1%
Between 1 and 5 minutes	**3**	27.3%
Between 5 and 10 minutes	**1**	9.1%
More than 10 minutes	**1**	9.1%
N/A	**5**	45.5%

Did you think the emergency team arrived fast enough?



Yes	**2**	18.2%
No	**4**	36.4%
N/A	**5**	45.5%

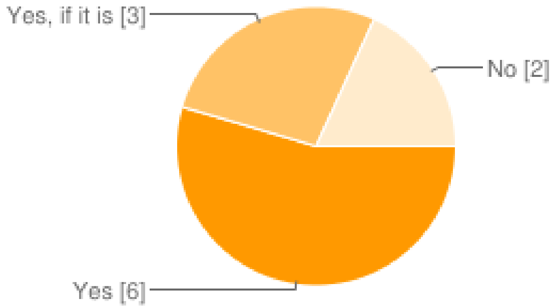
If your phone can automatically send your exact location to the call center, which would speed up the process and will allow the emergency team to head in the right direction earlier, would you be happy to use it?



Yes	**8**	72.7%
No	**1**	9.1%
Only if I have to manually agree to send my location, during the call.	**2**	18.2%

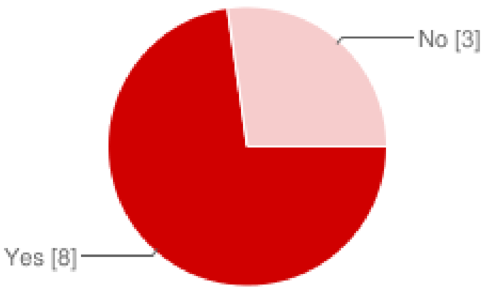
If your phone could video contact the call center via an application and stream real time video or pictures that would ease the process of describing

what has happened and will allow the emergency team to arrive with the correct number of people and equipment, would you be happy to use it?



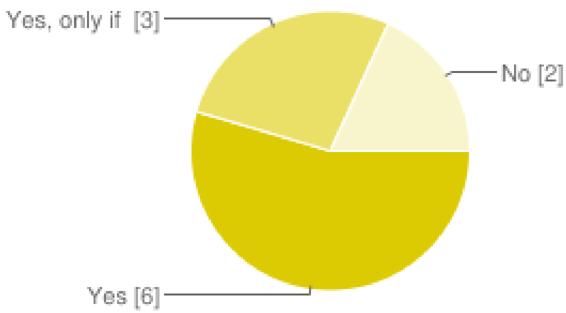
Yes	**6**	54.5%
Yes, if it is free	**3**	27.3%
No	**2**	18.2%

Have you heard of drones (also known as quadrocopters)?



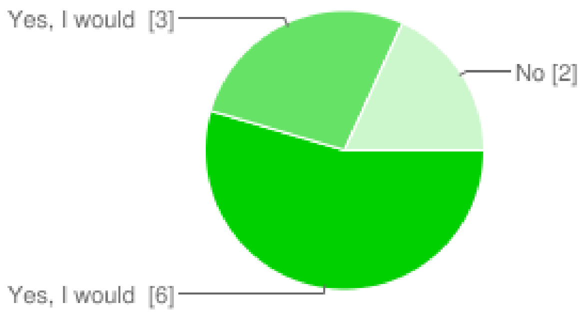
Yes	**8**	72.7%
No	**3**	27.3%

Would you feel comfortable if in an event of an accident, a drone with equipment is being sent by the emergency team to help you?



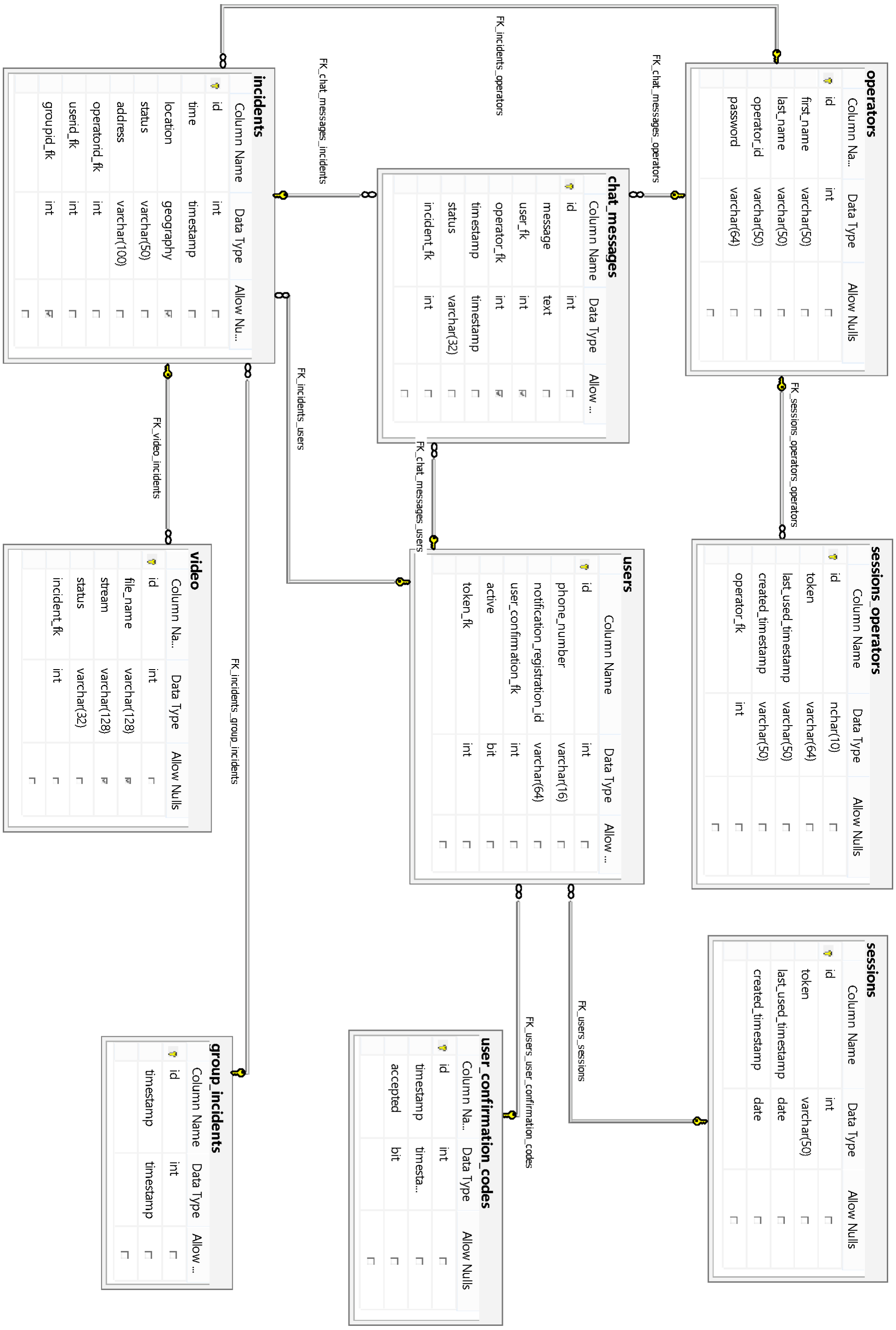
Yes	**6**	54.5%
Yes, only if it is safe enough	**3**	27.3%
No	**2**	18.2%

Would you like to be able to see the progress of the emergency team, coming to you?



Yes, I would like to be able to see exactly where they are	**6**	54.5%
Yes, I would like to know if they are on the move or busy with another call.	**3**	27.3%
No	**2**	18.2%

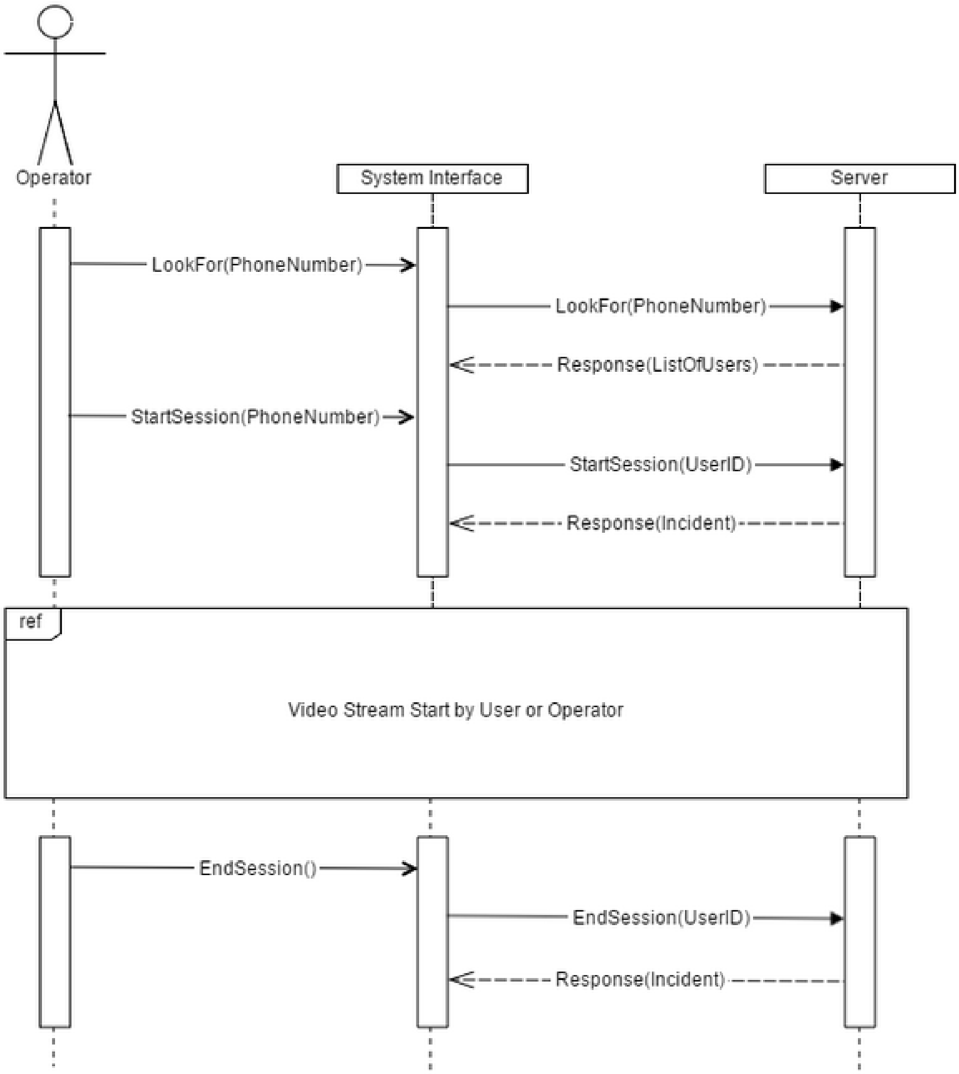
7.2 Emergency App Database



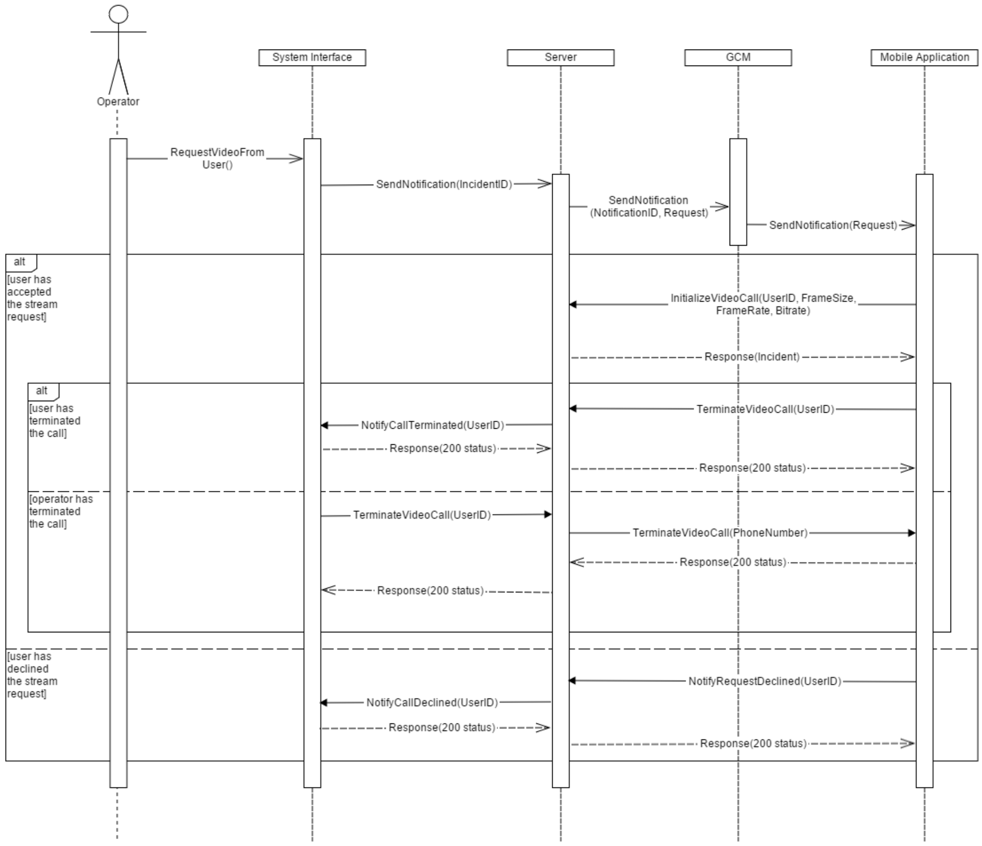
Database for the Emergency App

7.3 Video Stream

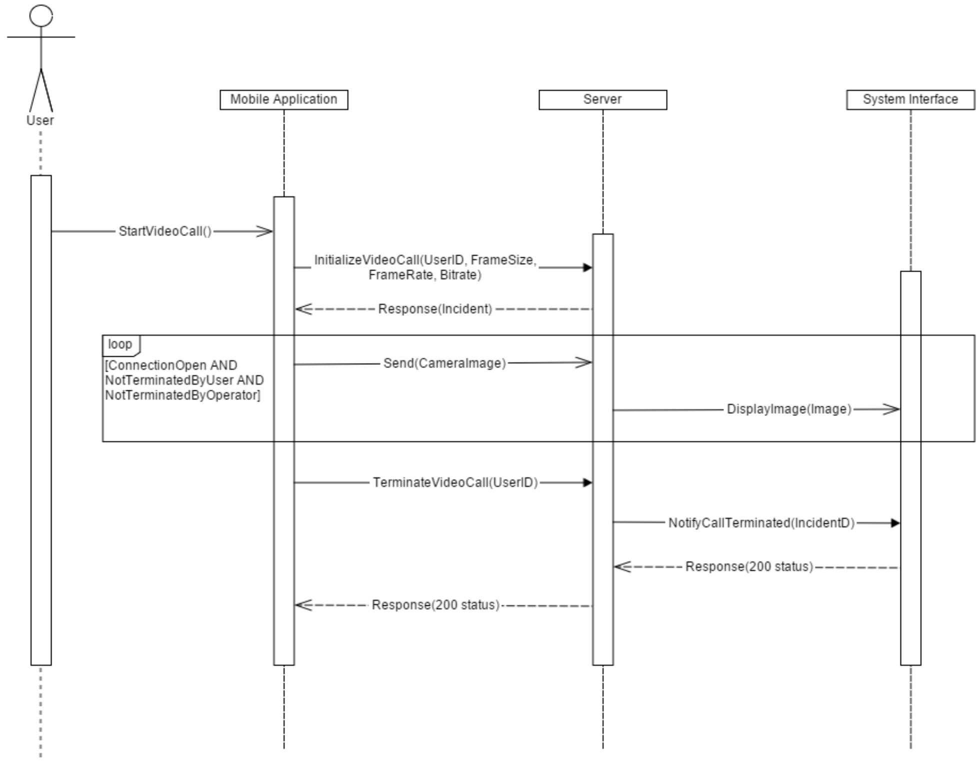
7.3.0.1 Iteration 1 Appendices for iteration 1.

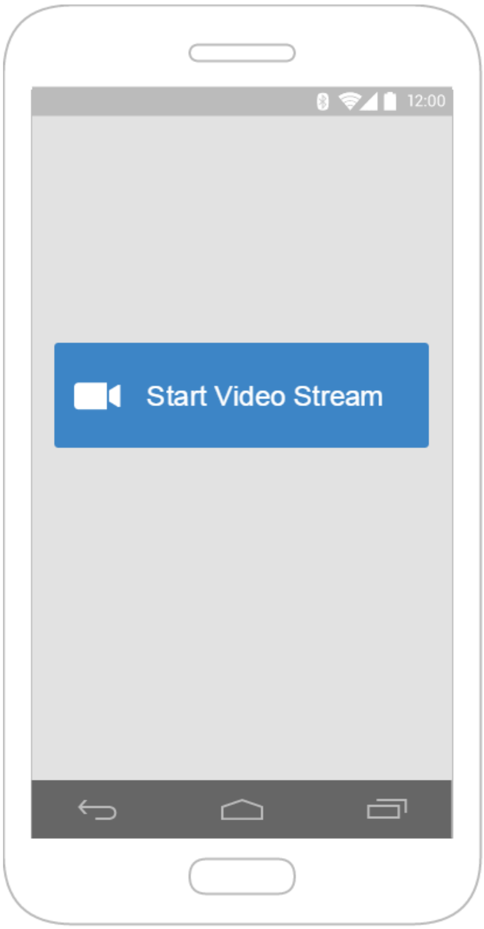


Sequence diagram for creating an Incident by the operator:

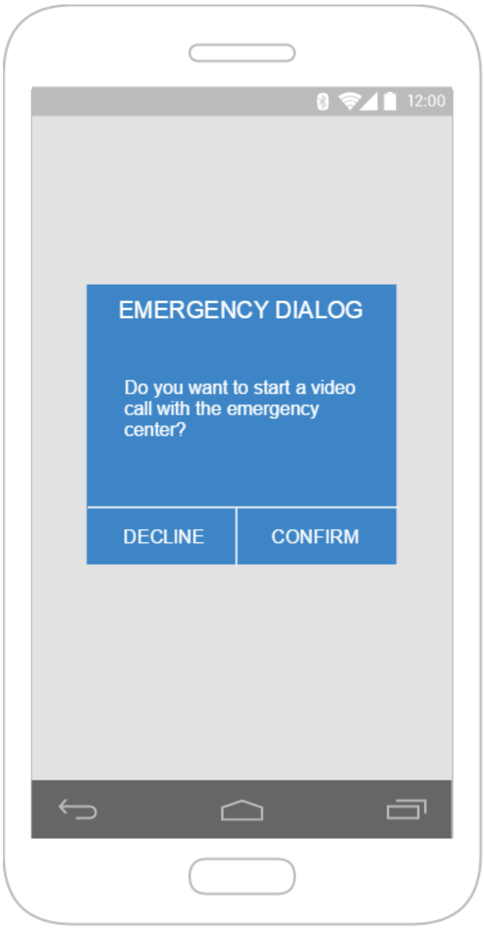


Sequence diagram for requesting the start of a video stream by the operator:

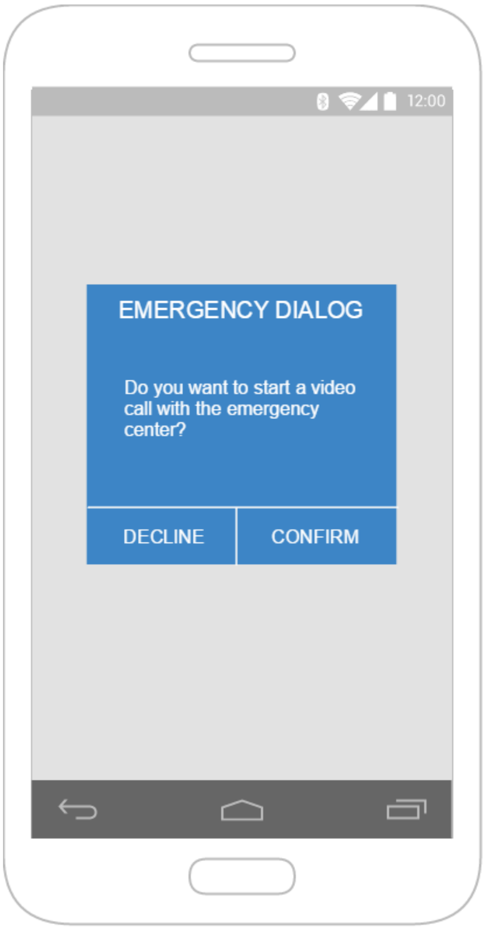

User
Mobile Application
Server
System Interface
StartVideoCall()
InitializeVideoCall(UserID, FrameSize, FrameRate, Bitrate)
Response(Incident)
loop
[ConnectionOpen AND NotTerminatedByUser AND NotTerminatedByOperator]
Send(CameraImage)
DisplayImage(Image)
TerminateVideoCall(UserID)
NotifyCallTerminated(IncidentD)
Response(200 status)
Response(200 status)



Sequence diagram for starting the video stream by the user:



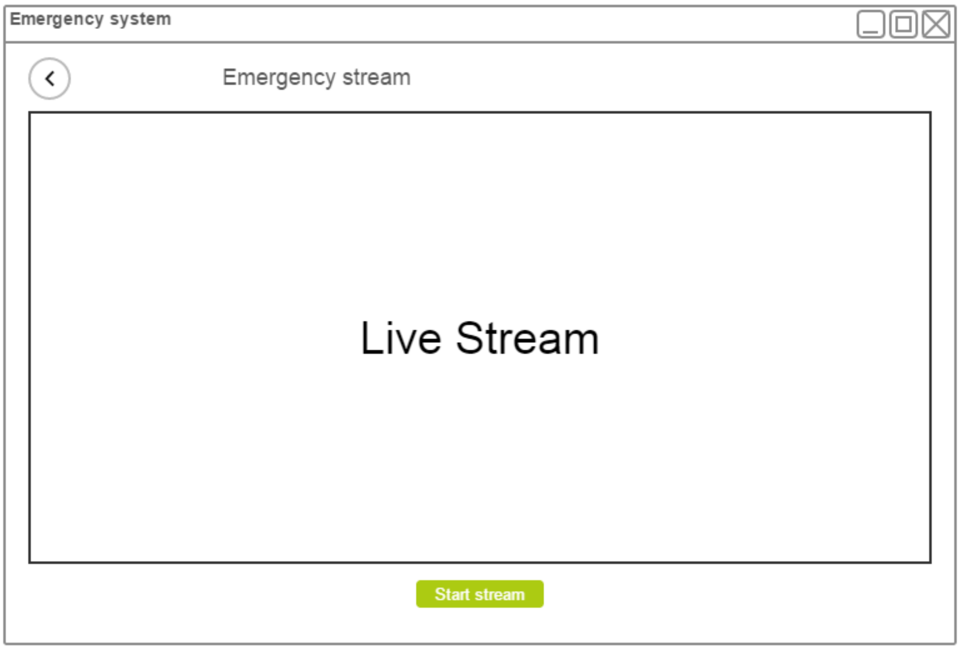
Mobile application screen showing the confirm dialog for starting the video stream:



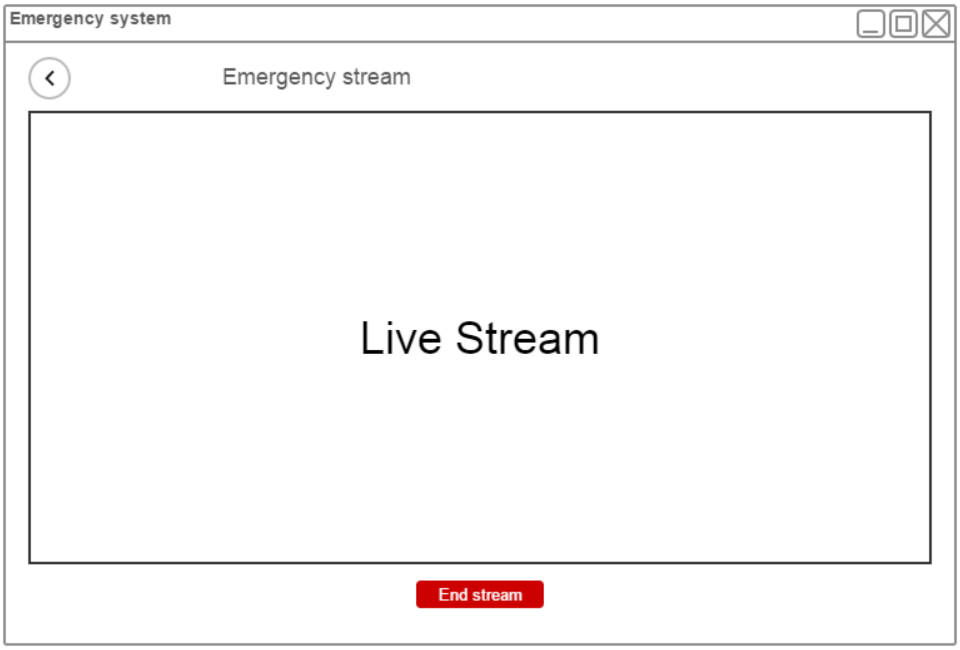
Sequence diagram for starting the video stream by the user:

Mobile application screen showing the confirm dialog for starting the video stream:

UI for the operator, providing functionality of searching for a phone number, in order to start an Incident session.



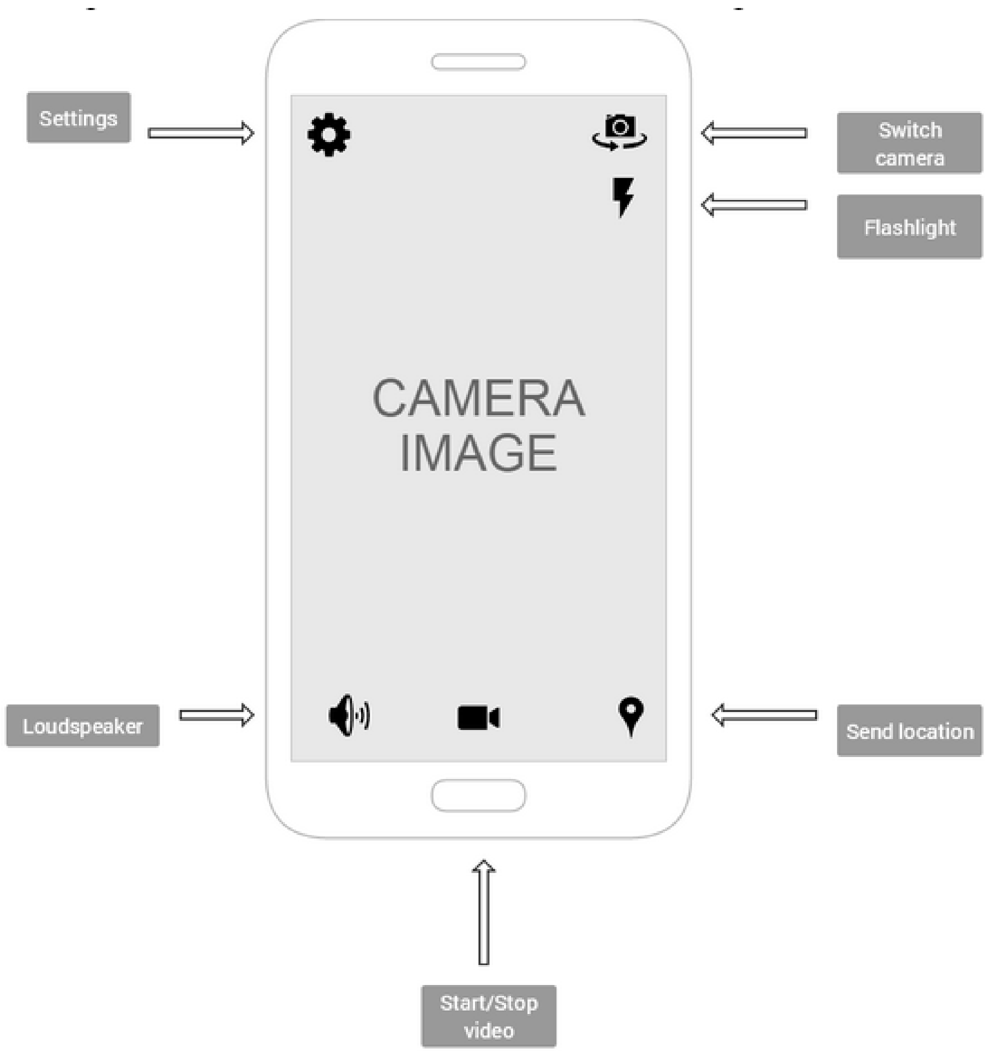
UI for the operator to request the start of a video stream, once a session has been initiated.



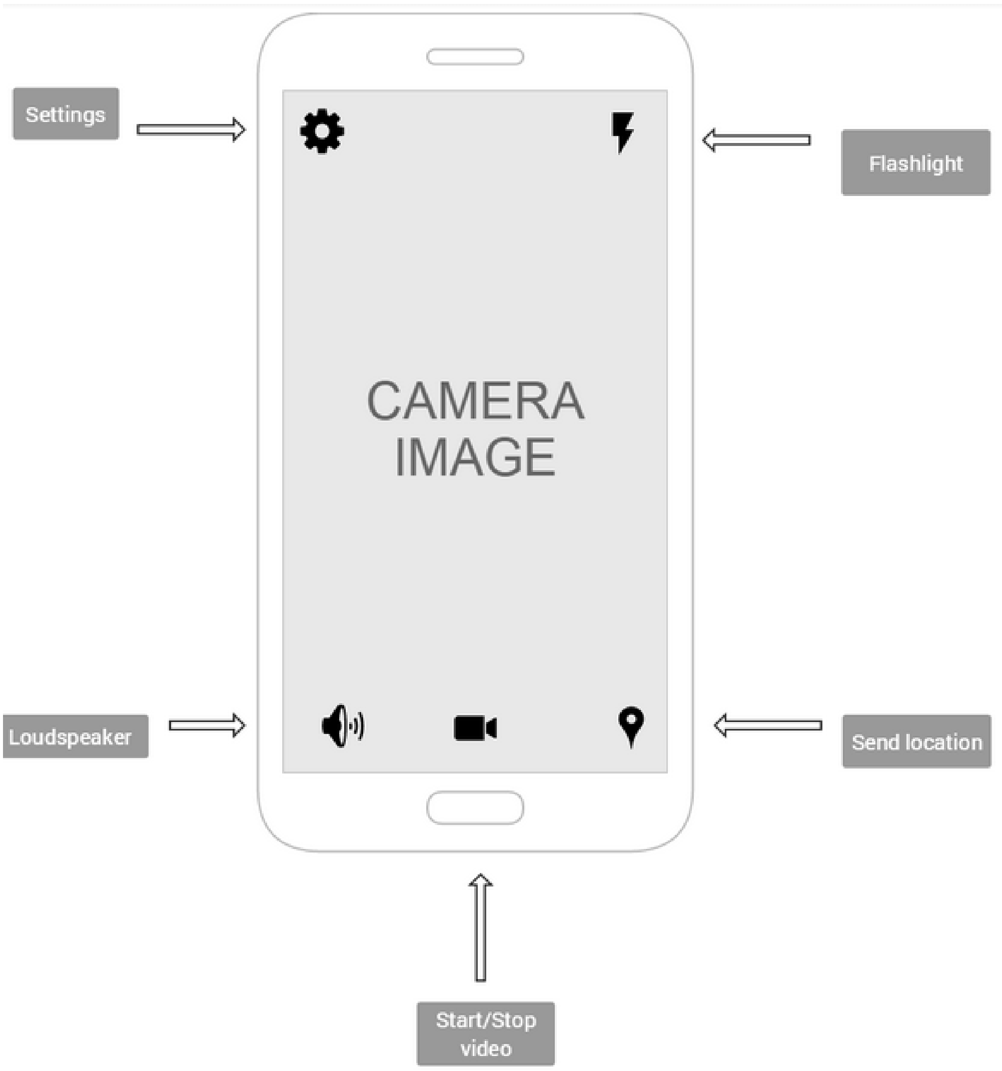
UI for the operator, in order to see the live stream and to stop the stream:

7.3.1 Iteration 2

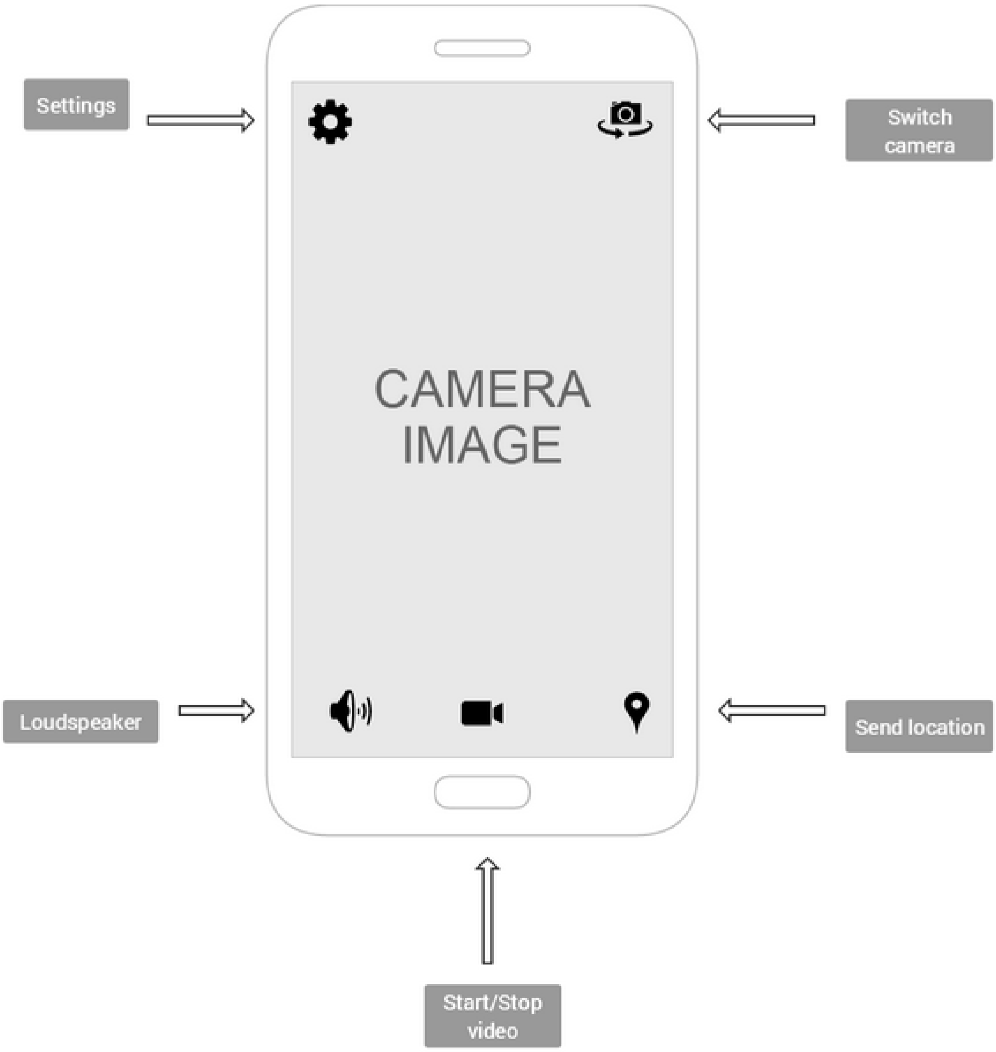
Appendices for iteration 2.



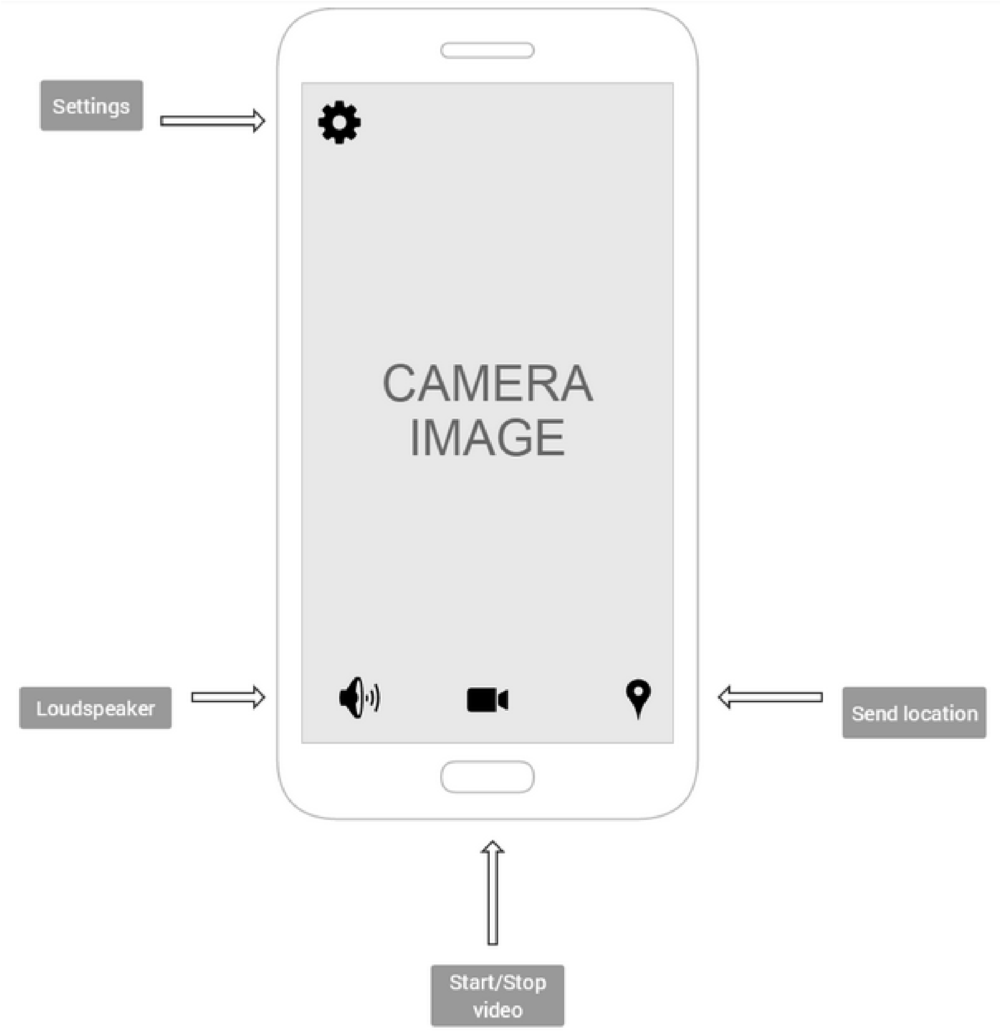
UI during video stream for devices with front-camera and flashlight.



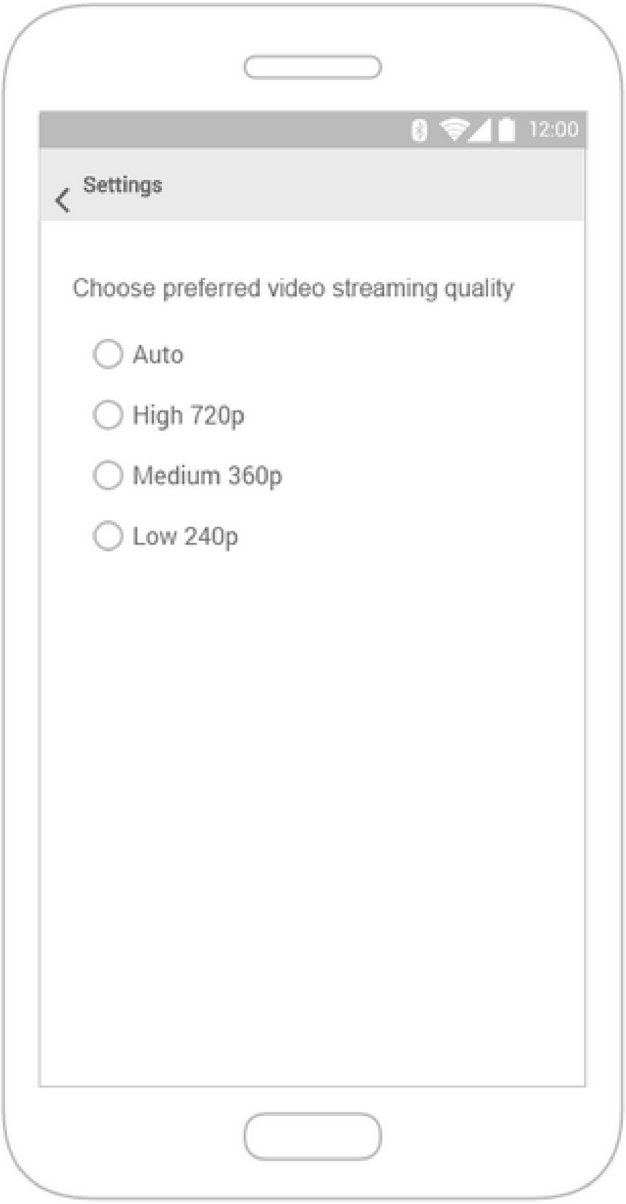
UI during video stream for devices without front-camera but with flashlight.



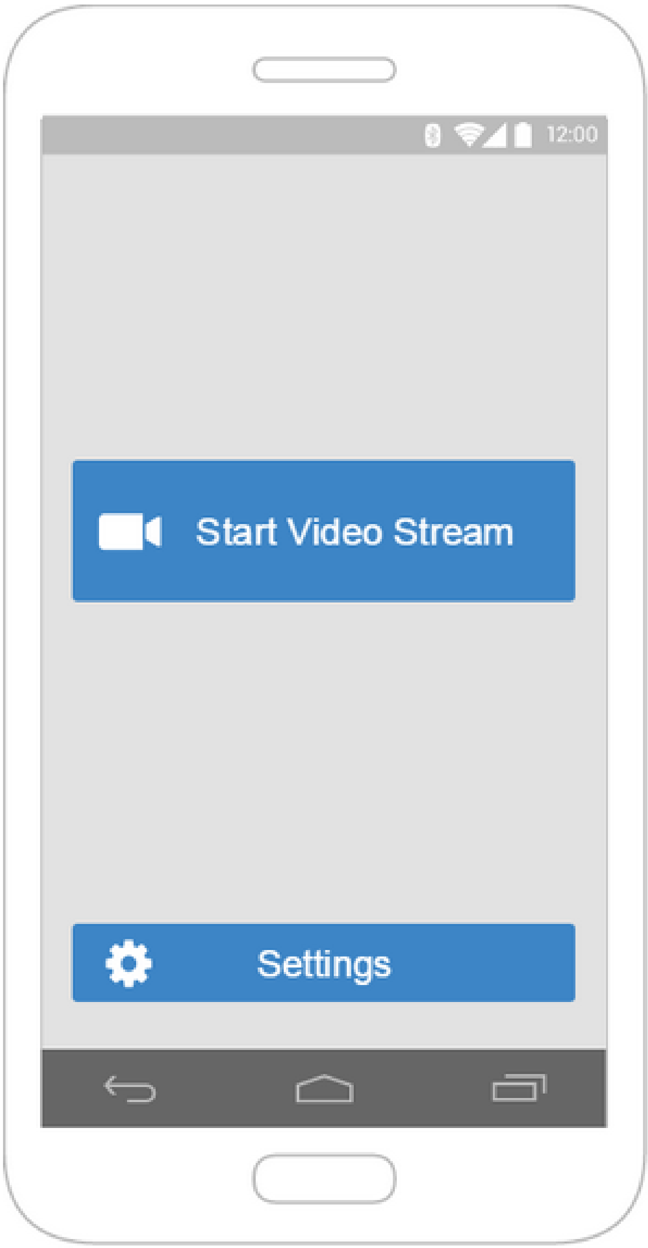
UI during video stream for devices with front-camera but without flashlight.



UI during video stream for devices without front-camera and flashlight.



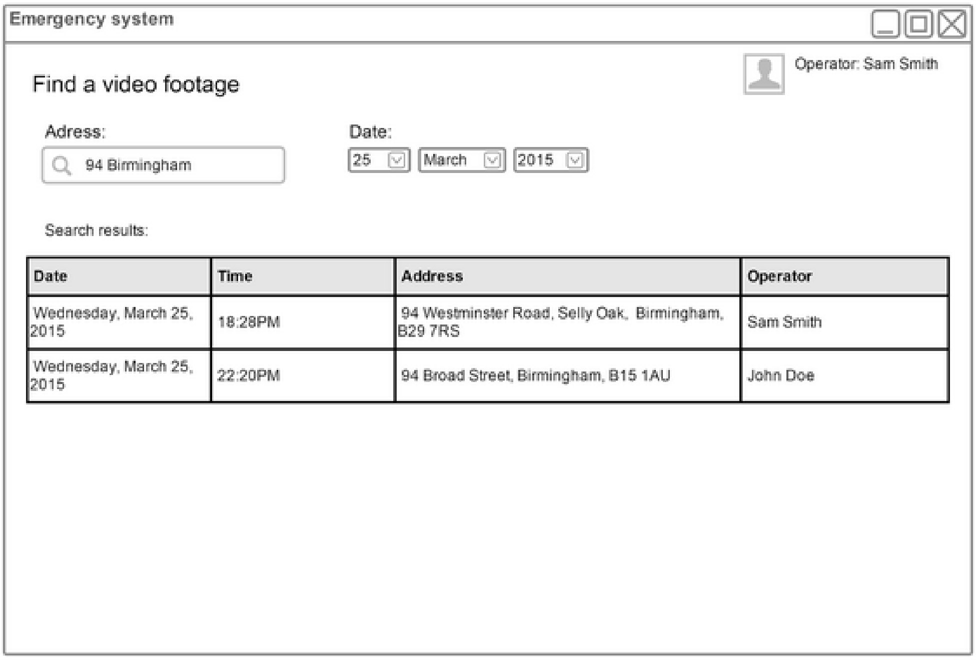
UI for setting the prefered video streaming quality during a video stream. Also available from main menu screen.



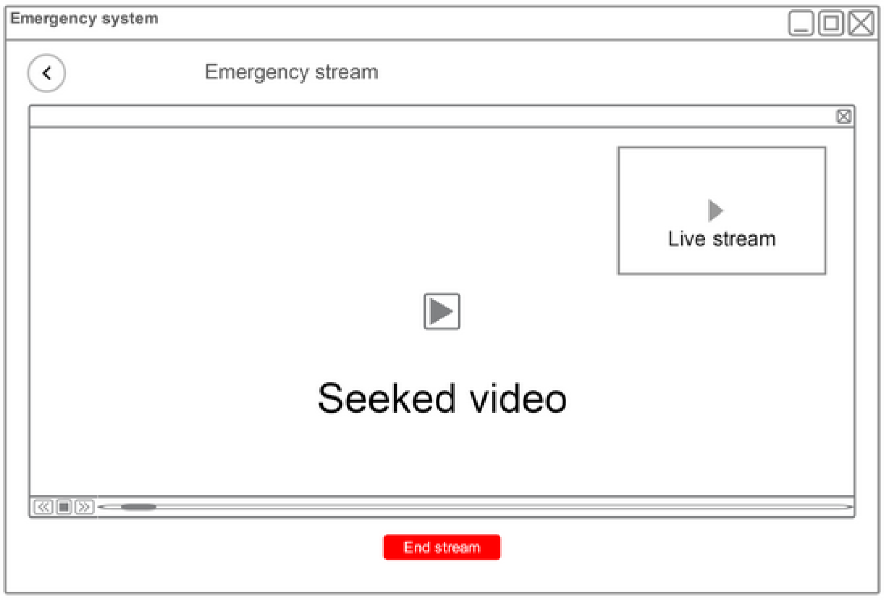
Main Menu screen of the mobile app:

7.3.2 Iteration 3

Appendices for iteration 3.



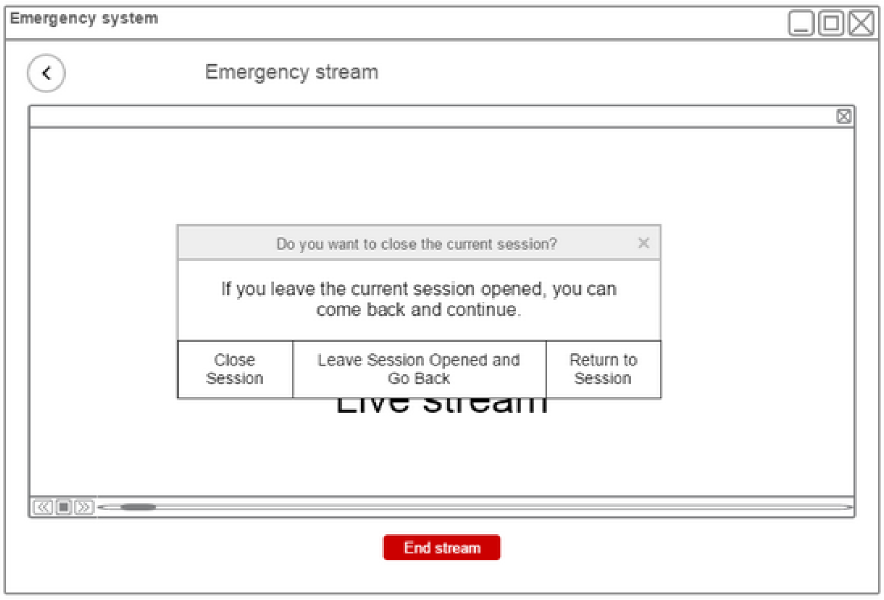
Backend for searching old incidents:



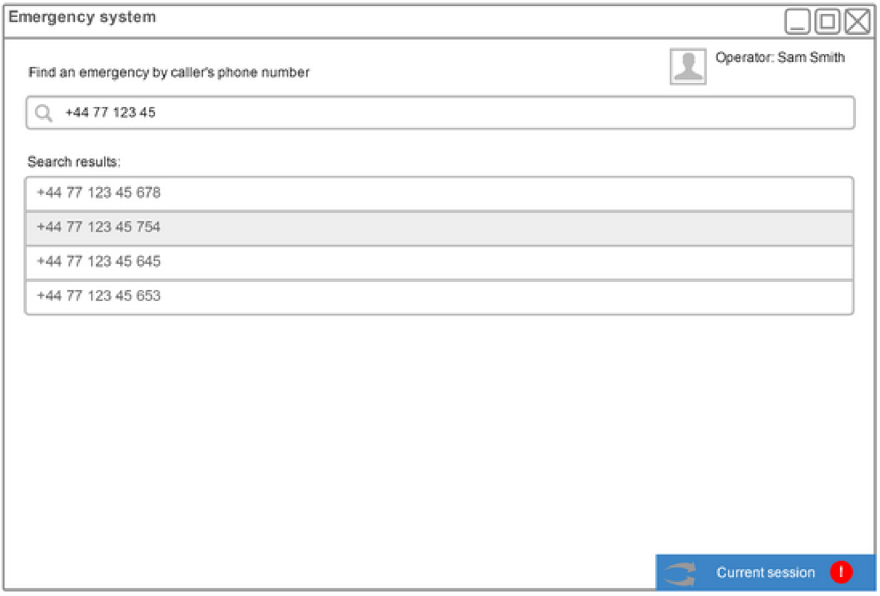
UI for the operator to skip/seek video:

7.3.3 Iteration 4

Appendices for iteration 4.



Confirmation dialog on back press, asking for closing the incident session, leaving it open or going back to the screen:



UI screen to show a currently opened session:

e

7.3.4 Iteration 4

Appendices for iteration 4.

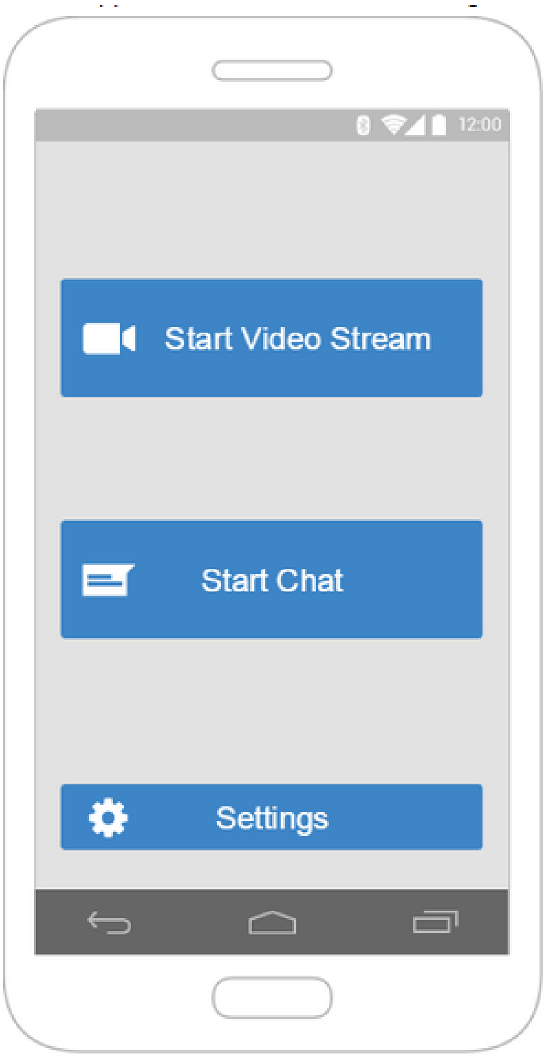
UI to show that more video streams are available for the current incident:

UI to show current available streams for the current incident.

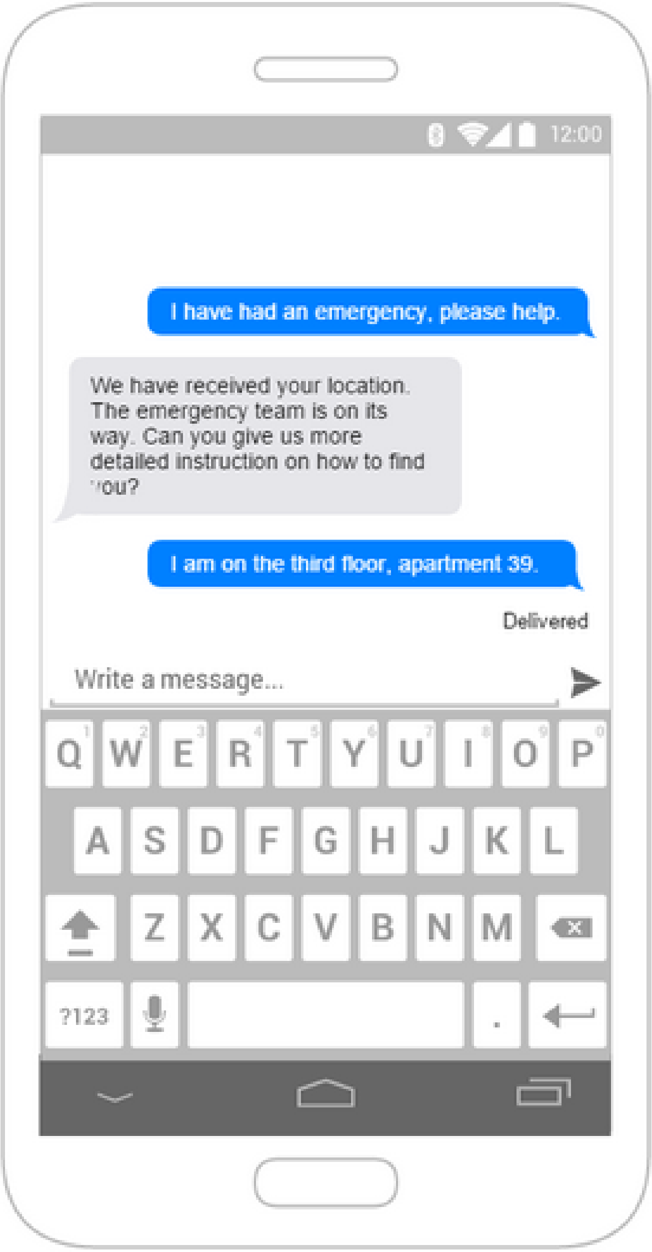
7.4 Chat

7.4.1 Iteration 1

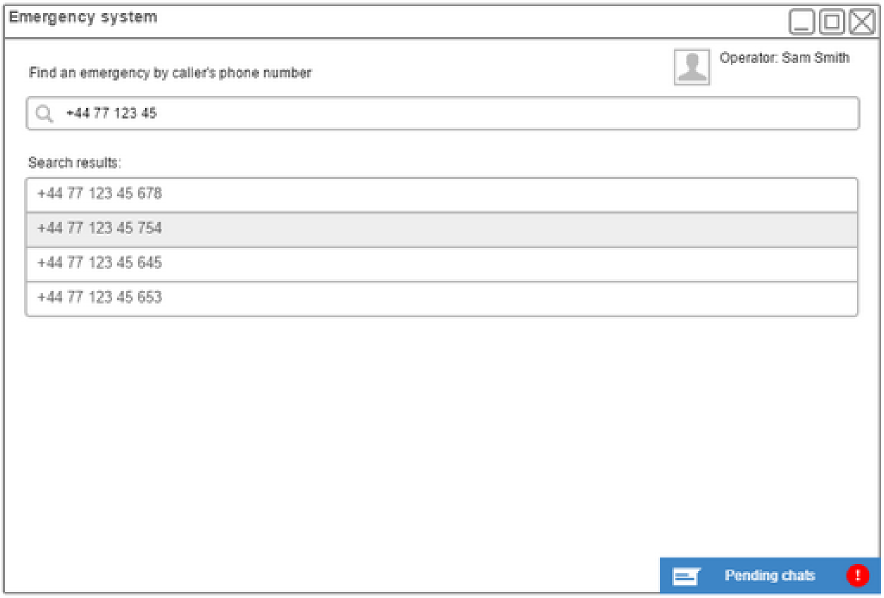
Appendices for iteration 1.



Mobile application Main Menu for starting a chat:

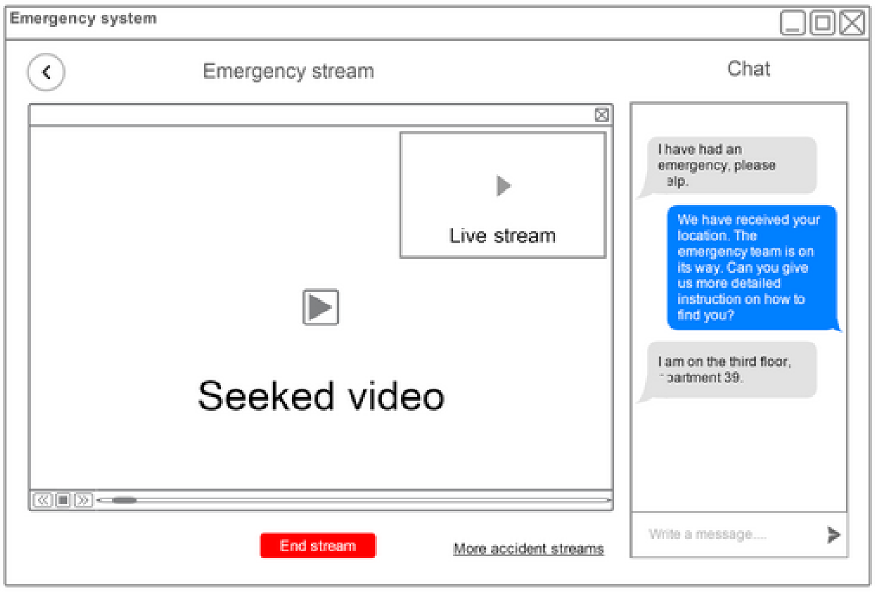


Mobile application UI for the live chat:

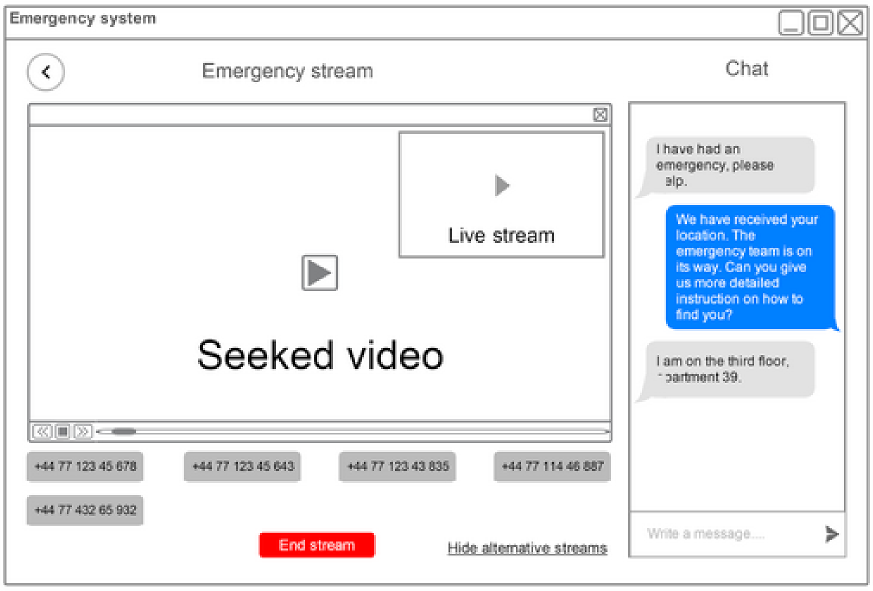


Backend UI for showing currently pending chats:

UI for showing chat in an active incident session:



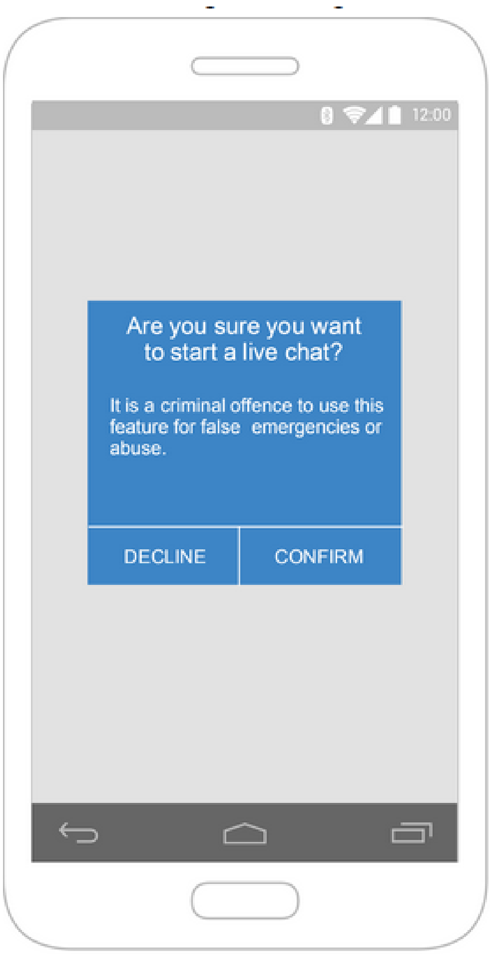
UI for Chat with multiple video streams available:



UI for Chat with multiple video streams available:

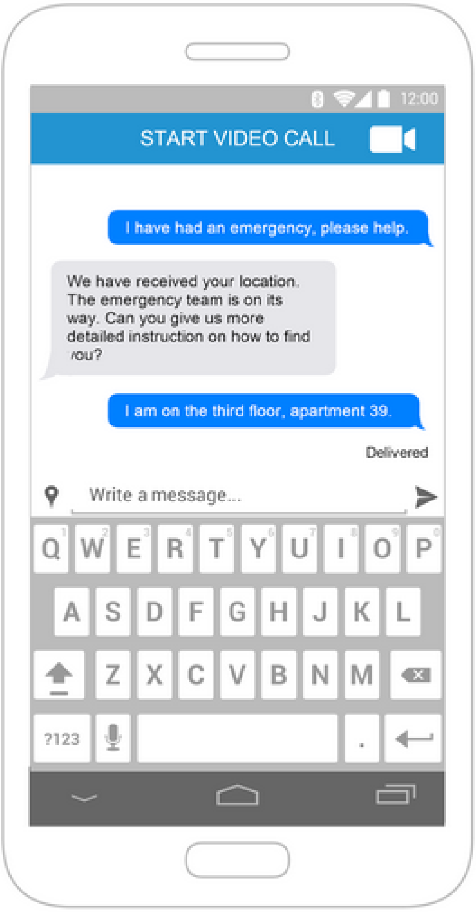
7.4.2 Iteration 2

Appendices for iteration 2.

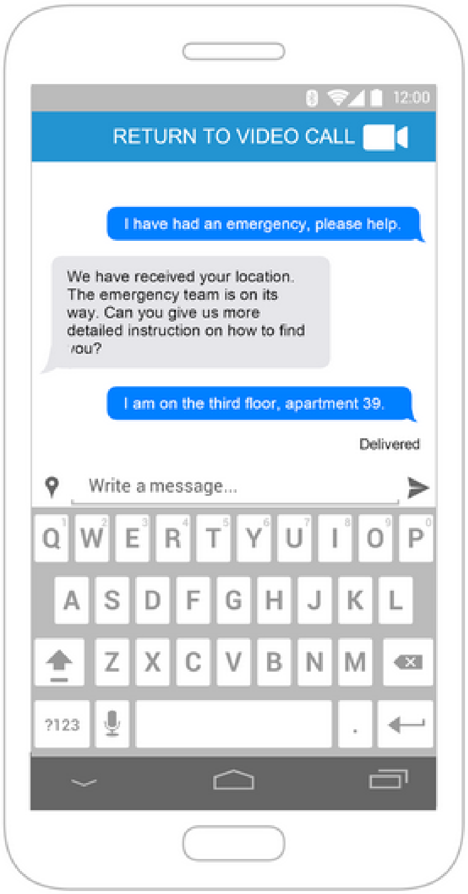


Confirmation dialog for starting a live chat:

Video stream UI for switching to chat and new messages indicator:



The chat screen with a button for starting a video stream and sending the location:

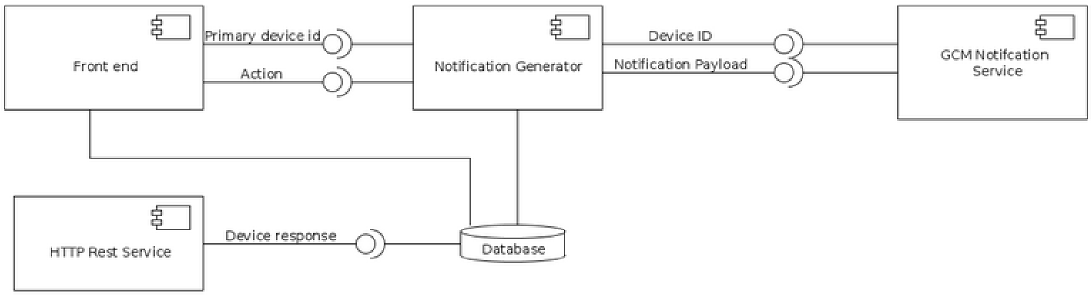


Video stream UI for switching to chat and new messages indicator:

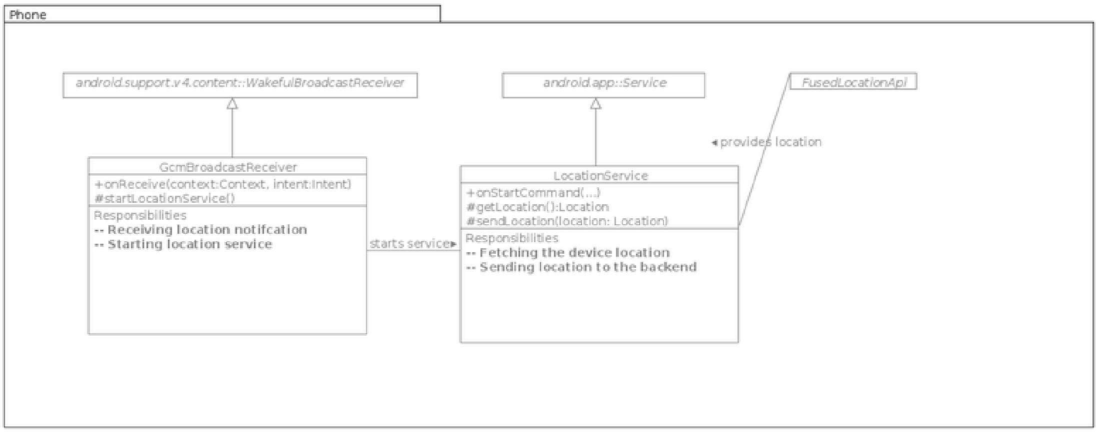
7.5 Automatic Video Sending

7.5.1 Iteration 1

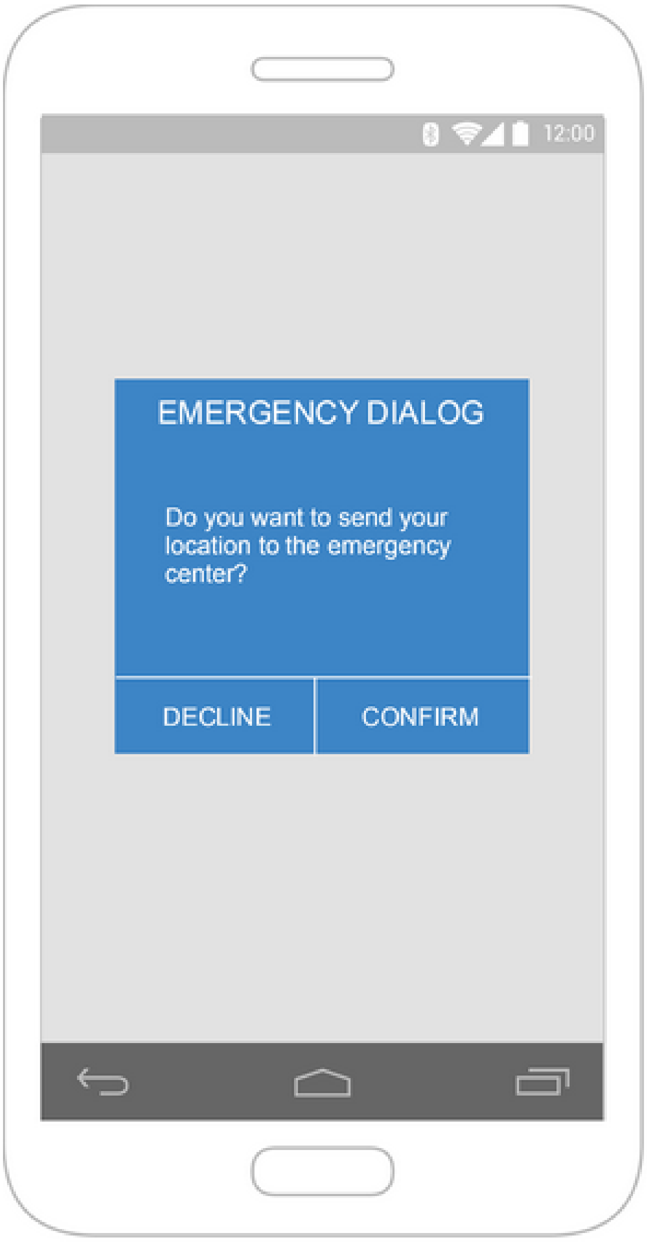
Appendices for iteration 1.



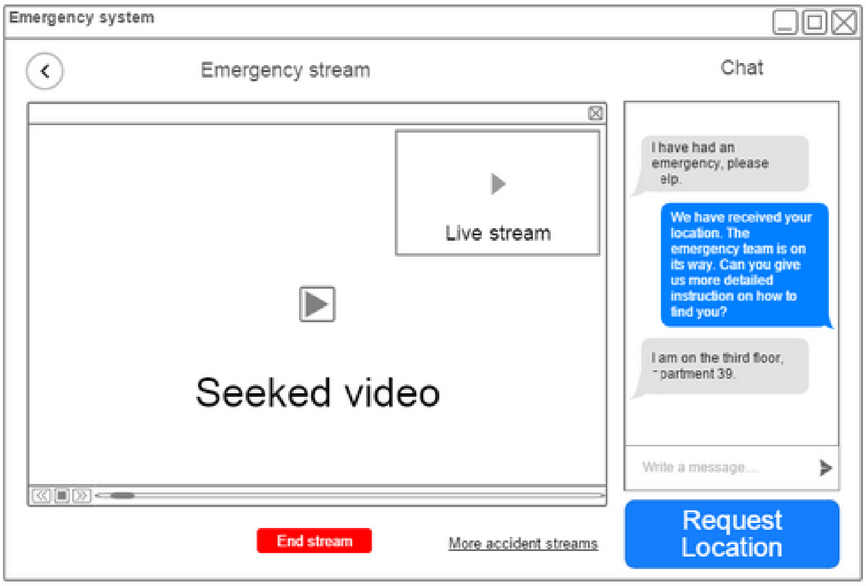
Deployment Diagram:



Class diagram:



Mobile application confirm dialog for sending the current location:



Request location button in the backend UI:

Location received on backend UI:

7.6 API for Client & Server

Emergency API

API for location and video call mobile applications

Version 1.0.0

Paths List all paths

/users/register

POST /users/register

Users

Summary

Register as regular user of the app

Description

Endpoint for the users of the video call and the location services

Parameters

Name	Located in	Description	Required	Schema
body	body	The registration details of the user	Yes	⇄ ▾**UserRegisterBody** { notificationRegistrationId: ▾ string phoneNumber: ▾ string }

Responses

Code	Description	Schema
200	Response with user id	⇄ ▾**UserRegistrationResponse** { id: ▾ string }
default	Unexpected error	⇄ ▾**Error** { code: integer fields: string message: string }

Try this operation

/users/{user-id}/activate

PUT /users/{user-id}/activate

Users

Summary

Activate a user and receive auth token

Description

Endpoint for activating newly registered user. The endpoint requires an activation key received by sms or other means

Parameters

Name	Located in	Description	Required	Schema
user-id	path	The registration details of the user	Yes	⇄ string
body	body	Confirmation details	Yes	⇄ ▾ActivateBody { confirmationCode: ▾ string }

Responses

Code	Description	Schema
200	Token used to identify the user	⇄ ▾AuthorizationResponse { token: ▾ string }
default	Unexpected error	⇄ ▾Error { code: integer fields: string message: string }

Try this operation

/incidents/{incident-id}/location

PUT /incidents/{incident-id}/location

incidents

Summary

Send the location of an incident

Description

Endpoint sending a location for an incident

Parameters

Name	Located in	Description	Required	Schema
incident-id	path	The id of the incident	Yes	⇄ string
body	body	Location details	Yes	⇄ ▾Location { latitude: number longitude: number }
token	header	Authorization token identifiyng the user	Yes	⇄ string

Responses

Code	Description		Schema
200	Token used to identify the user	⇄	▾**incident** { id: ▾ string location: ▾**Location** { latitude: number longitude: number } status: ▾**incidentStatus** string **Enum:** ▾ Object 0: "accepted" 1: "pendingChat" 2: "proccessing" 3: "processed" timestamp: string }

Try this operation

/incidents/

GET /incidents/

incidents

Summary

Get incidents

Description

Get incidents associated with the current user with status accepted or processing

Parameters

Name	Located in	Description	Required	Schema
token	header	Authorization token identifiyng the user	Yes	⇄ string

Responses

Code	Description	Schema
200	Token used to identify the user	⇄

```
▾ [
  ▾incident {
    id:         ▾ string
    location:   ▾Location {
                  latitude:  number
                  longitude: number
                }
    status:     ▾incidentStatus string
                  Enum:
                    ▾ Object
                      0: "accepted"
                      1: "pendingChat"
                      2: "proccessing"
                      3: "processed"
    timestamp: string
  }
]
```

Try this operation

Models List all models

- ActivateBody

```
▾ActivateBody {
⇄   confirmationCode: ▾ string
}
```

- AuthorizationResponse

```
▾AuthorizationResponse {
⇄   token: ▾ string
}
```

- Error

```
▾Error {
    code:    integer
⇄   fields:  string
    message: string
}
```

- Location

```
▾Location {
    latitude:  number
⇄   longitude: number
}
```

- UserRegisterBody

```
▾UserRegisterBody {
    notificationRegistrationId: ▾ string
⇄   phoneNumber:                ▾ string
}
```

- UserRegistrationResponse

```
▾UserRegistrationResponse {
⇄   id: ▾ string
}
```

- incident

```
▾incident {
    id:        ▾ string
    location:  ▾Location {
                  latitude:  number
                  longitude: number
               }
    status:    ▾incidentStatus string
```

```
⇄                    Enum:
                       ▾ Object
                         0: "accepted"
                         1: "pendingChat"
                         2: "proccessing"
                         3: "processed"
      timestamp: string
    }
```

- incidentStatus

```
    ▾ incidentStatus string
      Enum:
        ▾ Object
⇄         0: "accepted"
          1: "pendingChat"
          2: "proccessing"
          3: "processed"
```

Emergency API

API for location and video backend

Version 1.0.0

Paths List all paths

`/operators/register` List all operations

`POST /operators/register`

Operators

Summary

Register as an operator

Description

Register as an operator of the backend structure

Parameters

Name	Located in	Description	Required	Schema
body	body	The registration details of the user	Yes	⇄ ▾**OperatorRegisterBody** { firstName: ▾ string; lastName: ▾ string; operatorIdentifier: ▾ string; password: string }

Responses

Code	Description	Schema
200	The newly registered operator	⇄ ▾**OperatorRegisterResponse** { authorizationToken: ▾ **AuthorizationResponse** { token: ▾ string }; firstName: ▾ string; lastname: ▾ string; operatorIdentifierId: ▾ string }
default	Unexpected error	⇄ ▾**Error** { code: integer; fields: string; message: string }

Try this operation

/operators/login

POST /operators/login

Operators

Summary

Login as an operator

Description

Endpoint to login as an operator and receive authorization token

Parameters

Name	Located in	Description	Required	Schema
body	body	Loging credentials of the operator	No	⇄ ▾OperatorLoginBody { operatorIdentifier: ▾ string password: string }

Responses

Code	Description	Schema
200	The token used to sign all consequitive requests	⇄ ▾AuthorizationResponse { token: ▾ string }
default	Unexpected error	⇄ ▾Error { code: integer fields: string message: string }

Try this operation

/users/search/

GET /users/search/

Users

Summary

Get user details

Description

Get the details of the user by phone number

Parameters

Name	Located in	Description	Required	Schema
token	header		No	⇄ string
userPhoneNumber	query		No	⇄ string

Responses

Code	Description	Schema
200	The details of the user	⇄ ▾ [▾**UserDetailsResponse {** id: string phoneNumber: string }]
default	Unexpected error	⇄ ▾**Error {** code: integer fields: string message: string }

Try this operation

POST /users/{user-id}/incidents

Users Incindents

Summary

Create an incident for the user

Description

Create and incident associated with the user

Parameters

Name	Located in	Description	Required	Schema
user-id	path		No	⇄ string
token	header		No	⇄ string
body	body	Optional information regarding the accident	No	⇄

```
▾DetailedIncidentInfo {
  additionalDetails: string
  age:               number
  bleeding:          boolean
  breathing:         boolean
  chestPain:         boolean
  gender:            ▾Gender string
                       Enum:
                         ▾ Object
                           0: "male
                           "
                           1: "fema
                           le"
  medicalHistory:    string
  patientState:      ▾PatientState
                     string
                       Enum:
                         ▾ Object
                           0: "male
                           "
                           1: "fema
                           le"
}
```

Responses

Code	Description	Schema

```
▾Incident {
  detailedInfo: ▾DetailedIncidentInfo {
                  additionalDetails: string
                  age:               number
                  bleeding:          boolean
                  breathing:         boolean
                  chestPain:         boolean
                  gender:            ▾Gender string
```

201	The token used to sign all consequitive requests	⇄	<pre> Enum:
			                      ▾ Object
			                        0: "male
			                        "
			                        1: "fema
			                        le"
			    medicalHistory: string
			    patientState: ▾PatientState
			                       string
			                         Enum:
			                           ▾ Object
			                             0: "male
			                             "
			                             1: "fema
			                             le"
			  }
			id: ▾ string
			location: ▾Location {
			                latitude: number
			                longitude: number
			             }
			status: ▾incidentStatus string
			                Enum:
			                  ▾ Object
			                    0: "accepted"
			                    1: "pendingChat"
			                    2: "proccessing"
			                    3: "processed"
			timestamp: string
			videoStream: ▾Video {
			                status: ▾VideoStatus string
			                          Enum:
			                            ▾ Object
			                              0: "accepted"
			                              1: "pending"
			                              2: "declined"
			                stream: string
			             }
			}</pre>

default	Unexpected error	⇄	<pre>▾Error {
			  code: integer
			  fields: string
			  message: string
			}</pre>

Try this operation

/incidents/{incident-id}/location

POST /incidents/{incident-id}/location

Users Incindents

Summary

Location Request

Description

Issue location request from the device

Parameters

Name	Located in	Description	Required	Schema
incident-id	path		No	⇄ string
token	header		No	⇄ string

Responses

Code	Description	Schema
200	The token used to sign all consequitive requests	⇄ (see schema below)

```
▾Incident {
  detailedInfo:  ▾DetailedIncidentInfo {
                   additionalDetails: string
                   age:               number
                   bleeding:          boolean
                   breathing:         boolean
                   chestPain:         boolean
                   gender:            ▾Gender string
                                        Enum:
                                          ▾ Object
                                          0: "male
                                          "
                                          1: "fema
                                          le"
                   medicalHistory:    string
                   patientState:      ▾PatientState
                                      string
                                        Enum:
                                          ▾ Object
                                          0: "male
                                          "
                                          1: "fema
                                          le"
                 }
  id:            ▾ string
  location:      ▾Location {
                   latitude:  number
                   longitude: number
                 }
  status:        ▾incidentStatus string
                   Enum:
                     ▾ Object
                     0: "accepted"
                     1: "pendingChat"
```

```
                                2: "proccessing"
                                3: "processed"
    timestamp:    string
    videoStream:  ▾Video {
                     status: ▾VideoStatus string
                               Enum:
                                 ▾ Object
                                   0: "accepted"
                                   1: "pending"
                                   2: "declined"
                     stream: string
                  }
}
```

default	Unexpected error	⇄	▾**Error** { code: integer, fields: string, message: string }

```
▾Error {
   code:    integer
   fields:  string
   message: string
}
```

Try this operation

/incidents/{incident-id}/request-people

POST /incidents/{incident-id}/request-people

CPR Incindents

Summary

Request help

Description

Request help from registered CPR users

Parameters

Name	Located in	Description	Required	Schema
incident-id	path		No	⇄ string
token	header		No	⇄ string

Responses

Code	Description	Schema
200	The token used to sign all consequitive requests	⇄ (see schema below)

```
▾Incident {
  detailedInfo: ▾DetailedIncidentInfo {
                  additionalDetails: string
                  age:               number
                  bleeding:          boolean
                  breathing:         boolean
                  chestPain:         boolean
                  gender:            ▾Gender string
                                       Enum:
                                         ▾ Object
                                           0: "male
                                           "
                                           1: "fema
                                           le"
                  medicalHistory:    string
                  patientState:      ▾PatientState
                                     string
                                       Enum:
                                         ▾ Object
                                           0: "male
                                           "
                                           1: "fema
                                           le"
                }
  id:           ▾ string
  location:     ▾Location {
                  latitude:  number
                  longitude: number
                }
  status:       ▾incidentStatus string
                  Enum:
                    ▾ Object
                      0: "accepted"
                      1: "pendingChat"
```

```
                                2: "proccessing"
                                3: "processed"
        timestamp:    string
        videoStream:  ▾Video {
                        status: ▾VideoStatus string
                                 Enum:
                                  ▾ Object
                                   0: "accepted"
                                   1: "pending"
                                   2: "declined"
                        stream: string
                      }
      }
```

default	Unexpected error	⇄	▾**Error {** code: integer fields: string message: string }

Try this operation

/incidents/{incident-id}/video-stream

POST /incidents/{incident-id}/video-stream

Users Incindents Video

Summary

Video Stream Request

Description

Issue video stream request from the device

Parameters

Name	Located in	Description	Required	Schema
incident-id	path		No	⇄ string
token	header		No	⇄ string

Responses

Code	Description		Schema
200	The token used to sign all consequitive requests	⇄	(see below)

```
▾Incident {
  detailedInfo: ▾DetailedIncidentInfo {
                  additionalDetails: string
                  age:               number
                  bleeding:          boolean
                  breathing:         boolean
                  chestPain:         boolean
                  gender:            ▾Gender string
                                       Enum:
                                         ▾ Object
                                         0: "male
                                         "
                                         1: "fema
                                         le"
                  medicalHistory:    string
                  patientState:      ▾PatientState
                                     string
                                       Enum:
                                         ▾ Object
                                         0: "male
                                         "
                                         1: "fema
                                         le"
                }
  id:           ▾ string
  location:     ▾Location {
                  latitude:  number
                  longitude: number
                }
  status:       ▾incidentStatus string
                  Enum:
                    ▾ Object
                    0: "accepted"
                    1: "pendingChat"
```

```
                                        2: "proccessing"
                                        3: "processed"
              timestamp:    string
              videoStream:  ▾Video {
                              status: ▾VideoStatus string
                                        Enum:
                                          ▾ Object
                                            0: "accepted"
                                            1: "pending"
                                            2: "declined"
                              stream: string
                            }
            }
```

default	Unexpected error	⇄	▾**Error {** code: integer fields: string message: string }

Try this operation

/incidents/{incident-id}/video-stream1

DELETE /incidents/{incident-id}/video-stream1

Users Incindents Video

Summary

Terminate Video Stream

Description

Force the video Stream to be terminated

Parameters

Name	Located in	Description	Required	Schema
incident-id	path		No	⇄ string
token	header		No	⇄ string

Responses

Code	Description	Schema
200	The token used to sign all consequitive requests	⇄

```
▾Incident {
  detailedInfo: ▾DetailedIncidentInfo {
                  additionalDetails: string
                  age:               number
                  bleeding:          boolean
                  breathing:         boolean
                  chestPain:         boolean
                  gender:            ▾Gender string
                                       Enum:
                                         ▾ Object
                                           0: "male
                                           "
                                           1: "fema
                                           le"
                  medicalHistory:    string
                  patientState:      ▾PatientState
                                     string
                                       Enum:
                                         ▾ Object
                                           0: "male
                                           "
                                           1: "fema
                                           le"
                }
  id:           ▾ string
  location:     ▾Location {
                  latitude:  number
                  longitude: number
                }
  status:       ▾incidentStatus string
                  Enum:
                    ▾ Object
                      0: "accepted"
                      1: "pendingChat"
```

```
                  2: "proccessing"
                  3: "processed"
   timestamp:    string
   videoStream:  ▾Video {
                    status: ▾VideoStatus string
                              Enum:
                               ▾ Object
                                 0: "accepted"
                                 1: "pending"
                                 2: "declined"
                    stream: string
                 }
}
```

default	Unexpected error	⇄	▾**Error {** code: integer fields: string message: string }

Try this operation

/incidents/{incident-id}/

GET /incidents/{incident-id}/

Users Incindents

Summary

Get incident details

Description

Get details

Parameters

Name	Located in	Description	Required	Schema
incident-id	path		No	⇄ string
token	header		No	⇄ string

Responses

Code	Description	Schema
200	success	⇄

```
▾Incident {
  detailedInfo: ▾DetailedIncidentInfo {
                  additionalDetails: string
                  age:               number
                  bleeding:          boolean
                  breathing:         boolean
                  chestPain:         boolean
                  gender:            ▾Gender string
                                       Enum:
                                         ▾ Object
                                           0: "male
                                           "
                                           1: "fema
                                           le"
                  medicalHistory:    string
                  patientState:      ▾PatientState
                                     string
                                       Enum:
                                         ▾ Object
                                           0: "male
                                           "
                                           1: "fema
                                           le"
                }
  id:           ▾ string
  location:     ▾Location {
                  latitude:  number
                  longitude: number
                }
  status:       ▾incidentStatus string
                  Enum:
                    ▾ Object
                      0: "accepted"
                      1: "pendingChat"
```

```
                2: "proccessing"
                3: "processed"
  timestamp:    string
  videoStream:  ▾Video {
                  status: ▾VideoStatus string
                          Enum:
                            ▾ Object
                            0: "accepted"
                            1: "pending"
                            2: "declined"
                  stream: string
                }
}
```

default	Unexpected error	⇄	▾**Error** {
			code: integer
			fields: string
			message: string
			}

Try this operation

PUT /incidents/{incident-id}/1

Users Incindents

Summary

Change incident info

Description

Change incident details/status etc

Parameters

Name	Located in	Description	Required	Schema
incident-id	path		No	⇄ string
token	header		No	⇄ string
body	body		Yes	⇄

```
▾Incident {
  detailedInfo: ▾DetailedIncidentInfo {
                   additionalDetails: str
                   age:               num
                   bleeding:          boo
                   breathing:         boo
                   chestPain:         boo
                   gender:            ▾Ge
                                        E
                   medicalHistory:    str
                   patientState:      ▾Pa
                                      str
                                        E
                 }
  id:            ▾ string
  location:      ▾Location {
                   latitude:  number
                   longitude: number
                 }
  status:        ▾incidentStatus string
                  Enum:
                    ▾ Object
                      0: "accepted"
                      1: "pendingChat"
                      2: "proccessing"
                      3: "processed"
```

```
timestamp:    string
videoStream:  ▾Video {
                 status: ▾VideoStatus s
                          Enum:
                            ▾ Object
                             0: "acce
                             1: "pend
                             2: "decl
                 stream: string
              }
        }
```

Responses

Code	Description	Schema
200	success	⇄ (see schema below)

```
▾Incident {
  detailedInfo: ▾DetailedIncidentInfo {
                   additionalDetails: string
                   age:               number
                   bleeding:          boolean
                   breathing:         boolean
                   chestPain:         boolean
                   gender:            ▾Gender string
                                        Enum:
                                          ▾ Object
                                           0: "male
                                           "
                                           1: "fema
                                           le"
                   medicalHistory:    string
                   patientState:      ▾PatientState
                                      string
                                        Enum:
                                          ▾ Object
                                           0: "male
                                           "
                                           1: "fema
                                           le"
                }
  id:           ▾ string
  location:     ▾Location {
                   latitude:  number
                   longitude: number
                }
  status:       ▾incidentStatus string
                  Enum:
                    ▾ Object
                     0: "accepted"
                     1: "pendingChat"
                     2: "proccessing"
                     3: "processed"
  timestamp:    string
  videoStream:  ▾Video {
                   status: ▾VideoStatus string
                            Enum:
                              ▾ Object
                               0: "accepted"
```

```
                                          1: "pending"
                                          2: "declined"
                               stream: string
                             }
                  }
```

default	Unexpected error	⇄	▾**Error** {
			code: integer
			fields: string
			message: string
			}

Try this operation

/incidents/

GET /incidents/

Admin Incindents

Summary

Get all incidents

Description

Get details

Parameters

Name	Located in	Description	Required	Schema
incidentStatus	query	filter by incident status	No	⇄ string
admin-token	header	Special token used for admin operations	No	⇄ string

Responses

Code	Description	Schema
200	The token used to sign all consequitive requests	⇄

```
▾Incident {
  detailedInfo: ▾DetailedIncidentInfo {
                  additionalDetails: string
                  age:               number
                  bleeding:          boolean
                  breathing:         boolean
                  chestPain:         boolean
                  gender:            ▾Gender string
                                       Enum:
                                         ▾ Object
                                           0: "male
                                           "
                                           1: "fema
                                           le"
                  medicalHistory:    string
                  patientState:      ▾PatientState
                                     string
                                       Enum:
                                         ▾ Object
                                           0: "male
                                           "
                                           1: "fema
                                           le"
                }
  id:           ▾ string
  location:     ▾Location {
                  latitude:  number
                  longitude: number
                }
  status:       ▾incidentStatus string
```

```
                              Enum:
                                ▾ Object
                                  0: "accepted"
                                  1: "pendingChat"
                                  2: "proccessing"
                                  3: "processed"
        timestamp:    string
        videoStream:  ▾Video {
                         status: ▾VideoStatus string
                                  Enum:
                                    ▾ Object
                                      0: "accepted"
                                      1: "pending"
                                      2: "declined"
                         stream: string
                      }
      }
```

default	Unexpected error	⇄	▾**Error** {
			code: integer
			fields: string
			message: string
			}

Try this operation

POST /incidents/group

Admin Incindents

Summary

Get all incidents

Description

Get details

Parameters

Name	Located in	Description	Required	Schema
admin-token	header	Special token used for admin operations	No	⇄ string
body	body	incidents to merge	No	⇄ ▾ [string]

Responses

Code	Description	Schema
200	The token used to sign all consequitive requests	⇄ (see below)

```
▾Incident {
  detailedInfo:  ▾DetailedIncidentInfo {
                   additionalDetails: string
                   age:               number
                   bleeding:          boolean
                   breathing:         boolean
                   chestPain:         boolean
                   gender:            ▾Gender string
                                        Enum:
                                          ▾ Object
                                            0: "male"
                                            1: "female"
                   medicalHistory:    string
                   patientState:      ▾PatientState
                                      string
                                        Enum:
                                          ▾ Object
                                            0: "male"
                                            1: "female"
                 }
  id:            ▾ string
  location:      ▾Location {
                   latitude:  number
                   longitude: number
                 }
```

```
status:        ▾incidentStatus string
                 Enum:
                   ▾ Object
                     0: "accepted"
                     1: "pendingChat"
                     2: "proccessing"
                     3: "processed"
timestamp:     string
videoStream:   ▾Video {
                 status: ▾VideoStatus string
                           Enum:
                             ▾ Object
                               0: "accepted"
                               1: "pending"
                               2: "declined"
                 stream: string
               }
}
```

default	Unexpected error	⇄	▾**Error** { code: integer fields: string message: string }

Try this operation

/incidents/{incident-id}/chats

GET /incidents/{incident-id}/chats

Users Incindents Chat

Summary

Get chat history

Description

Get details

Parameters

Name	Located in	Description	Required	Schema
incident-id	path		No	⇄ string
token	header		No	⇄ string

Responses

Code	Description	Schema
200	success	⇄ ▾ [▾**ChatMessage {** authorId: string id: string incidentId: string timestamp: string }]
default	Unexpected error	⇄ ▾**Error {** code: integer fields: string message: string }

Try this operation

/incidents/chats

GET /incidents/chats

Users Incindents Chat

Summary

Get chats

Description

Get all incident ids with incident status - pending chat

Parameters

Name	Located in	Description	Required	Schema
token	header		No	⇄ string

Responses

Code	Description	Schema
200	success	⇄ ▾ [string]
default	Unexpected error	⇄ ▾ **Error** { code: integer fields: string message: string }

Try this operation

Models List all models

- AuthorizationResponse

```
▾AuthorizationResponse {
⇄   token: ▾ string
}
```

- ChatMessage

```
▾ChatMessage {
    authorId:    string
    id:          string
⇄
    incidentId:  string
    timestamp:   string
}
```

- DetailedIncidentInfo

```
▾DetailedIncidentInfo {
    additionalDetails: string
    age:               number
    bleeding:          boolean
    breathing:         boolean
    chestPain:         boolean
    gender:            ▾Gender string
                         Enum:
⇄                          ▾ Object
                           0: "male"
                           1: "female"
    medicalHistory:    string
    patientState:      ▾PatientState string
                         Enum:
                           ▾ Object
                           0: "male"
                           1: "female"
}
```

- Error

```
▾Error {
    code:    integer
⇄   fields:  string
    message: string
}
```

- Gender

```
▾Gender string
```

```
⇄    Enum:
       ► Object
```

- Incident

```
▾Incident {
  detailedInfo: ▾DetailedIncidentInfo {
                   additionalDetails: string
                   age:               number
                   bleeding:          boolean
                   breathing:         boolean
                   chestPain:         boolean
                   gender:            ▾Gender string
                                        Enum:
                                          ▾ Object
                                          0: "male"
                                          1: "female"
                   medicalHistory:    string
                   patientState:      ▾PatientState string
                                        Enum:
                                          ▾ Object
                                          0: "male"
                                          1: "female"
                 }
  id:           ▾ string
  location:     ▾Location {
⇄                  latitude:  number
                   longitude: number
                 }
  status:       ▾incidentStatus string
                  Enum:
                    ▾ Object
                    0: "accepted"
                    1: "pendingChat"
                    2: "proccessing"
                    3: "processed"
  timestamp:    string
  videoStream:  ▾Video {
                   status: ▾VideoStatus string
                             Enum:
                               ▾ Object
                               0: "accepted"
                               1: "pending"
                               2: "declined"
                   stream: string
                 }
```

```
    }
```

- Location

```
▾Location {
  latitude:  number
  longitude: number
}
```

- OperatorLoginBody

```
▾OperatorLoginBody {
  operatorIdentifier: ▾ string
  password:             string
}
```

- OperatorRegisterBody

```
▾OperatorRegisterBody {
  firstName:          ▾ string
  lastName:           ▾ string
  operatorIdentifier: ▾ string
  password:             string
}
```

- OperatorRegisterResponse

```
▾OperatorRegisterResponse {
  authorizationToken:     ▾AuthorizationResponse {
                            token: ▾ string
                          }
  firstName:              ▾ string
  lastname:               ▾ string
  operatorIdentifierId:   ▾ string
}
```

- PatientState

```
▾PatientState string
  Enum:
    ▾ Object
      0: "male"
      1: "female"
```

- UserDetailsResponse

```
▾UserDetailsResponse {
  id:          string
  phoneNumber: string
}
```

- Video

```
▾Video {
   status: ▾VideoStatus string
              Enum:
                ▾ Object
⇄                 0: "accepted"
                  1: "pending"
                  2: "declined"
   stream: string
}
```

- VideoStatus

```
▾VideoStatus string
   Enum:
     ▾ Object
⇄      0: "accepted"
       1: "pending"
       2: "declined"
```

- incidentStatus

```
▾incidentStatus string
   Enum:
     ▾ Object
⇄      0: "accepted"
       1: "pendingChat"
       2: "proccessing"
       3: "processed"
```

8 Bibliography

[1] S. Rogers,
'Mortality statistics: every cause of death in England and Wales, visualised — Datablog',
the Guardian, 2011. [Online].
Available: http://www.theguardian.com/news/datablog/2011/oct/28/mortality-statistics-causes-death-england-wales-2010.
[Accessed: 16- Feb- 2015]

[2] NHS England,
'Consensus Paper on Out-of-Hospital Cardiac Arrest in England',
NHS England, 2014 [Online].
Available: https://www.resus.org.uk/pages/OHCA_consensus_paper.pdf.
[Accessed: 16- Feb- 2015]

[3] D. Campbell,
'Chances of surviving cardiac arrest in England 'dire",
The Guardian, 2012 [Online].
Available: http://www.theguardian.com/society/2012/nov/19/chances-surviving-cardiac-arrest-dire.
[Accessed: 16- Feb- 2015]

[4] T. Lindner, E. Sreide, O. Nilsen and M. Wenche Torunn,
'Good outcome in every fourth resuscitation attempt is achievableAn Utstein template report from the Stavanger region', 2011 [Online].
Available: http://www.sahlgrenska.se/upload/SU/omrade_molndal/ambulans/Good%20outcome%20in%20every%20fourth%20resuscitation%20attempt%20is%20achievable.%20Stavanger.pdf.
[Accessed: 16- Feb- 2015]

[5] NHS England,
'Increase bystander initiated CPR by increasing proportion of population trained in CPR from 3.8m people to 5 million nationally', 2014 [Online].
Available: http://www.england.nhs.uk/wp-content/uploads/2014/02/pm-fs-3-2.pdf.
[Accessed: 16- Feb- 2015]

[6] Smslivraddare.se,
'Info om hjrtstopp — SMSLivrddare', 2015. [Online].
Available: http://www.smslivraddare.se/info-om-hjartstopp.
[Accessed: 16- Feb- 2015]

[7] Hscic.gov.uk,
'Find data - Health & Social Care Information Centre', 2015. [Online].
Available: http://www.hscic.gov.uk/catalogue/PUB11062.
[Accessed: 16- Feb- 2015]

[8] P. Stephenson,
'Text messages are saving Swedes from cardiac arrest',
Quartz, 2015. [Online].
Available: http://qz.com/138334/text-messages-are-saving-swedes-from-cardiac-arrest/.
[Accessed: 16- Feb- 2015]

[9] A. Sifferlin,
'How Swedens New Text Message Plan Is Saving Cardiac Arrest Victims — TIME.com',
TIME.com, 2015. [Online].
Available: http://healthland.time.com/2013/10/23/how-swedens-new-text-message-plan-is-saving-cardiac-arrest-victims/.
[Accessed: 16- Feb- 2015]

[10] Developer.android.com,
'Google Cloud Messaging — Android Developers', 2015. [Online].

Available: https://developer.android.com/google/gcm/index.html.
[Accessed: 19- Feb- 2015]

[11] G. drain,
'Google Cloud Messaging and less battery drain',
Stackoverflow.com, 2015. [Online].
Available: http://stackoverflow.com/a/12124927.
[Accessed: 19- Feb- 2015]

[12] Wenda.baba.io,
'Is GCM service reliable for large scale push notification?', 2015. [Online].
Available: http://wenda.baba.io/questions/1527378/is-gcm-service-reliable-for-large-scale-push-notification.html.
[Accessed: 19- Feb- 2015]

[13] Google Developers,
'Orchestrating iOS Push Notifications on Google Cloud Platform', 2015. [Online].
Available: https://cloud.google.com/developers/articles/ios-push-notifications/.
[Accessed: 19- Feb- 2015]

[14] Developer.android.com,
'Implementing GCM Client on Android — Android Developers', 2015. [Online].
Available: https://developer.android.com/google/gcm/client.html.
[Accessed: 21- Feb- 2015]

[15] Developer.android.com,
'Server Reference — Android Developers', 2015. [Online].
Available: https://developer.android.com/google/gcm/server-ref.html.
[Accessed: 04- Mar- 2015]

[16] Developer.android.com,
'LatLng — Android Developers', 2015. [Online].
Available: https://developer.android.com/reference/com/google/android/gms/maps/model/LatLng.html.
[Accessed: 04- Mar- 2015]

[17] Developer.android.com,
'Common Intents — Android Developers', 2015. [Online].
Available: https://developer.android.com/guide/components/intents-common.html.
[Accessed: 04- Mar- 2015]

[18] Google Developers,
'Markers', 2015. [Online].
Available: https://developers.google.com/maps/documentation/android/marker.
[Accessed: 04- Mar- 2015] 1

[19] Developer.android.com,
'Geocoder — Android Developers', 2015. [Online].
Available: http://developer.android.com/reference/android/location/Geocoder.html.
[Accessed: 07- Mar- 2015]

[20] Developer.android.com,
'Address — Android Developers', 2015. [Online].
Available: http://developer.android.com/reference/android/location/Address.html.
[Accessed: 07- Mar- 2015]

[21] Developer.android.com,
'Distance to Location — Android Developers', 2015. [Online].
Available: http://developer.android.com/reference/android/location/Location.html.
[Accessed: 07- Mar- 2015]

[22] Developer.android.com,
'Notifications — Android Developers', 2015. [Online].

Available: http://developer.android.com/design/patterns/notifications.html.
[Accessed: 07- Mar- 2015]

[23] Developer.android.com,
'Implementing Collapseable Messages — Android Developers', 2015. [Online].
Available: https://developer.android.com/google/gcm/server.html.
[Accessed: 08- Mar- 2015]

[24] BBC News,
'Ambulance service miss emergency response times', 2013 [Online].
Available: http://Ambulance service miss emergency response times.
[Accessed: 08- Mar- 2015]

[25] Developer.android.com,
'GCM Cloud Connection Server (XMPP) — Android Developers', 2015. [Online].
Available: http://developer.android.com/google/gcm/ccs.html.
[Accessed: 13- Mar- 2015]

[26] Stackoverflow.com,
'Why and how is push-notification (like GCM) battery efficient?', 2015. [Online].
Available: http://stackoverflow.com/questions/22663778/why-and-how-is-push-notification-like-gcm-battery-efficient.
[Accessed: 13- Mar- 2015]

[27] Java.dzone.com,
'Upstream Messages with Google Cloud Messaging — Javalobby', 2015. [Online].
Available: http://java.dzone.com/articles/upstream-messages-google-cloud?page=0,1.
[Accessed: 13- Mar- 2015]

[28] Developer.android.com,
'Getting the Last Known Location — Android Developers', 2015. [Online].
Available: https://developer.android.com/training/location/retrieve-current.html.
[Accessed: 13- Mar- 2015]

[29] Rightleftrightwrong.com,
'Right, Left, Right, Wrong! - Handedness Statistics', 2015. [Online].
Available: http://www.rightleftrightwrong.com/statistics.html.
[Accessed: 18- Mar- 2015]

[30] BBC News,
'Ambulance times slowest since 2011', 2014. [Online].
Available: http://www.bbc.co.uk/news/uk-wales-30631210.
[Accessed: 18- Mar- 2015]

[31] Londonambulance.nhs.uk,
'London Ambulance Service - What happens when you call 999', 2015. [Online].
Available: http://www.londonambulance.nhs.uk/calling_999/what_happens_when_you_call_999.aspx.
[Accessed: 18- Apr- 2015]

[32] Londonambulance.nhs.uk,
'London Ambulance Service - How we use your personal information', 2015. [Online].
Available: http://www.londonambulance.nhs.uk/talking_with_us/use_your_personal_information.aspx.
[Accessed: 27- Apr- 2015]

[33] TopTenREVIEWS,
'Video Calling Phones Review - TopTenREVIEWS', 2015. [Online].
Available: http://video-calling-phones-review.toptenreviews.com/.
[Accessed: 27- Apr- 2015]

[34] Wikipedia,
'Smartphones', 2015. [Online].
Available: http://en.wikipedia.org/wiki/Smartphone.
[Accessed: 27- Apr- 2015]

[35] Gsmarena.com,
'GSMArena.com - GSM phone reviews, news, opinions, votes, manuals and more...', 2015. [Online].
Available: http://www.gsmarena.com/.
[Accessed: 27- Apr- 2015]

[36] Mobilemastinfo.com,
'Stats and Facts', 2015. [Online].
Available: http://www.mobilemastinfo.com/stats-and-facts/.
[Accessed: 27- Apr- 2015]

[37] Wikipedia,
'5G', 2015. [Online].
Available: http://en.wikipedia.org/wiki/5G.
[Accessed: 27- Apr- 2015]

[38] Support.skype.com,
'How much bandwidth does Skype need?', 2015. [Online].
Available: https://support.skype.com/en/faq/FA1417/how-much-bandwidth-does-skype-need.
[Accessed: 27- Apr- 2015]

[39] Support.google.com,
'Live encoder settings, bit rates and resolutions - YouTube Help', 2015. [Online].
Available: https://support.google.com/youtube/answer/2853702?hl=en-GB.
[Accessed: 27- Apr- 2015]

[40] Wikipedia,
'Real-time Transport Protocol', 2015. [Online].
Available: http://en.wikipedia.org/wiki/Real-time_Transport_Protocol.
[Accessed: 27- Apr- 2015]

[41] G. Kelly,
'H.265 vs VP9: 4K video codecs explained', Trustedreviews.com, 2014. [Online].
Available: http://www.trustedreviews.com/opinions/h-265-vs-vp9-4k-video-codes-explained.
[Accessed: 27- Apr- 2015]

[42] Wikipedia,
'VP9', 2015. [Online].
Available: http://en.wikipedia.org/wiki/VP9.
[Accessed: 27- Apr- 2015]

[43] S. Shankland,
'Google urges fast adoption of VP9 video compression - CNET', CNET, 2013. [Online].
Available: http://www.cnet.com/news/google-urges-fast-adoption-of-vp9-video-compression/.
[Accessed: 27- Apr- 2015]

[44] R. Wingard,
'The Video Codec Fight: H.265 vs. VP9', Blog.euclidiq.com, 2014. [Online].
Available: http://blog.euclidiq.com/video-compression-blog/the-video-codec-fight-h.265-vs.-vp9.
[Accessed: 27- Apr- 2015]

[45] S. Formats,
'Supported Media Formats — Android Developers', Developer.android.com, 2015. [Online].
Available: http://developer.android.com/guide/appendix/media-formats.html.
[Accessed: 27- Apr- 2015]

[46] Ustream Support,
'Internet connection and recommended encoding settings', 2015. [Online].
Available: https://ustream.zendesk.com/entries/22962268-Internet-connection-and-recommended-encoding-settings.

[Accessed: 27- Apr- 2015]

[47] Wikipedia,
'WebM', 2015. [Online].
Available: http://en.wikipedia.org/wiki/WebM.
[Accessed: 27- Apr- 2015]

[48] Webmproject.org,
'The WebM Project', 2015. [Online].
Available: http://www.webmproject.org/.
[Accessed: 27- Apr- 2015]

[49] Emergencysms.org.uk,
'SMS to the emergency services — EmergencySMS', 2015. [Online].
Available: http://www.emergencysms.org.uk/.
[Accessed: 27- Apr- 2015]

[50] Wikipedia,
'Short Message Service', 2015. [Online].
Available: http://en.wikipedia.org/wiki/Short_Message_Service.
[Accessed: 27- Apr- 2015]

[51] Emergencysms.org.uk,
'How to send text — EmergencySMS', 2015. [Online].
Available: http://www.emergencysms.org.uk/how_to_send_text.php.
[Accessed: 27- Apr- 2015]

[52] Emergencysms.org.uk,
'News — EmergencySMS', 2012. [Online].
Available: http://www.emergencysms.org.uk/news.php.
[Accessed: 27- Apr- 2015]

[53] Safenetwork.org.uk,
'Deaf awareness — Identifying deafness — The Safe Network', 2011. [Online].
Available: http://www.safenetwork.org.uk/training_and_awareness/pages/deaf-awareness.aspx.
[Accessed: 27- Apr- 2015]

[54] Wikipedia,
'WebSocket', 2015. [Online].
Available: http://en.wikipedia.org/wiki/WebSocket.
[Accessed: 27- Apr- 2015]

[55] Findlaw.co.uk,
'Hoax 999 calls - Crime and Justice', 2015. [Online].
Available: http://www.findlaw.co.uk/law/criminal/crimes_a_z/500552.html.
[Accessed: 27- Apr- 2015]

[56] Developer.android.com,
'Notification — Android Developers', 2015. [Online].
Available: https://developer.android.com/reference/android/app/Notification.html.
[Accessed: 28- Apr- 2015]

[57] W3.org,
'Hypertext Transfer Protocol Specification', 2015. [Online].
Available: http://www.w3.org/Protocols/rfc2616/rfc2616.txt.
[Accessed: 29- Apr- 2015]

[58] P. Barber,
'Phill Barber's Blog: Client Side vs Server Side Session',
Phillbarber.blogspot.co.uk, 2014. [Online].
Available: http://phillbarber.blogspot.co.uk/2014/02/client-side-vs-server-side-session.html.
[Accessed: 29- Apr- 2015]

[59] The JSON Data Interchange Format,
1st ed. ECMA International, 2013 [Online].
Available: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.
[Accessed: 29- Apr- 2015]

[60] GitHub,
'swagger-api/swagger-spec', 2015. [Online].
Available: https://github.com/swagger-api/swagger-spec.
[Accessed: 29- Apr- 2015]

[61] N. Carey,
'Establishing Pedestrian Walking Speeds', 2005 [Online].
Available: http://www.westernite.org/datacollectionfund/2005/psu_ped_summary.pdf.
[Accessed: 24- Apr- 2015]

[62] Json-schema.org,
'JSON Schema - Documentation', 2015. [Online].
Available: http://json-schema.org/documentation.html.
[Accessed: 29- Apr- 2015]