Software Design Study

'Final Report'

UNIVERSITY OF BIRMINGHAM

Emergency Systems Team

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

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

April 29, 2015

Contents

1 Abstract

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

---

2 Introduction

---

3 System Overview

---

4 Modules

4.1 Emergency App

4.2 CPR System

4.2.1 Rationale

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

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

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

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

So few recoverable cardiac arrests are survived mainly due to the time it takes between the arrest occurring and the rescue attempt beginning. With the average waiting time currently around 8 minutes [7], and recent news of longer

response times from emergency services [24], we need to look into ways of helping those affected before emergency services arrive.

4.2.2 Project SMS-livrddare

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

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

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

4.3 API

---

5 Specification

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

6 Common Technologies

6.1 Google Cloud Messaging

6.1.1 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.

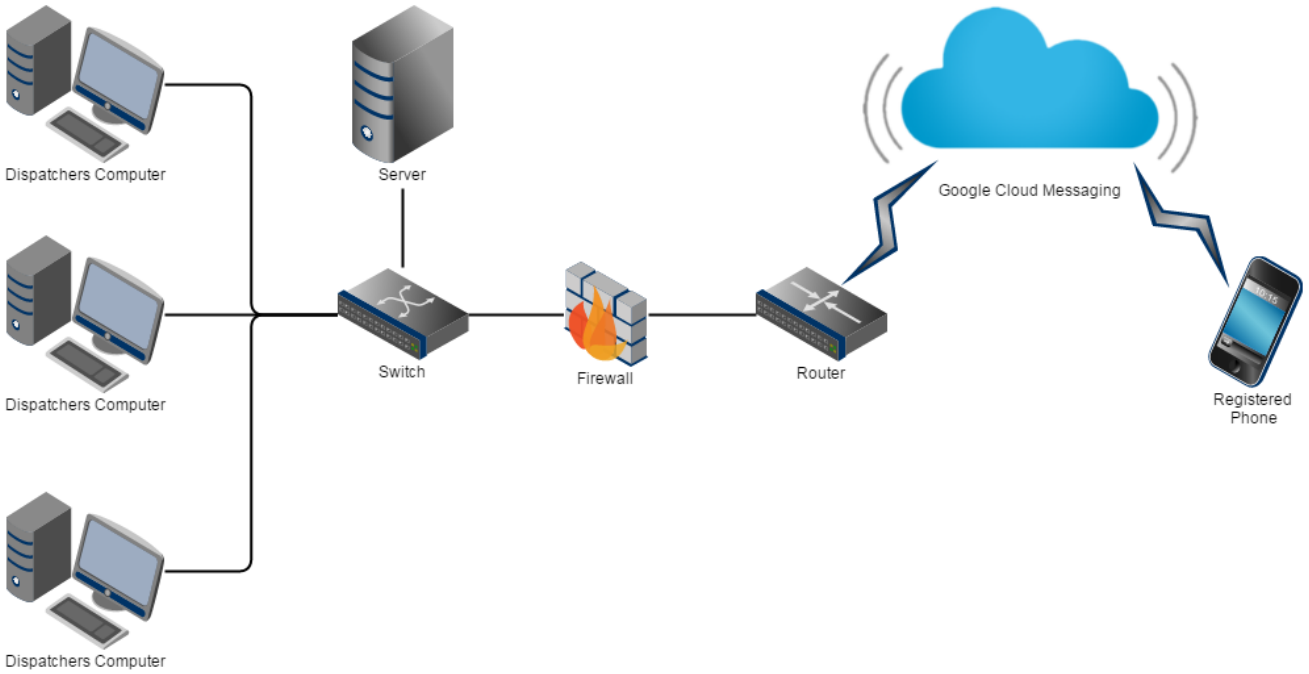

Figure 1: Network map of how our system will communicate between devices and the server

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.

6.1.2 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].

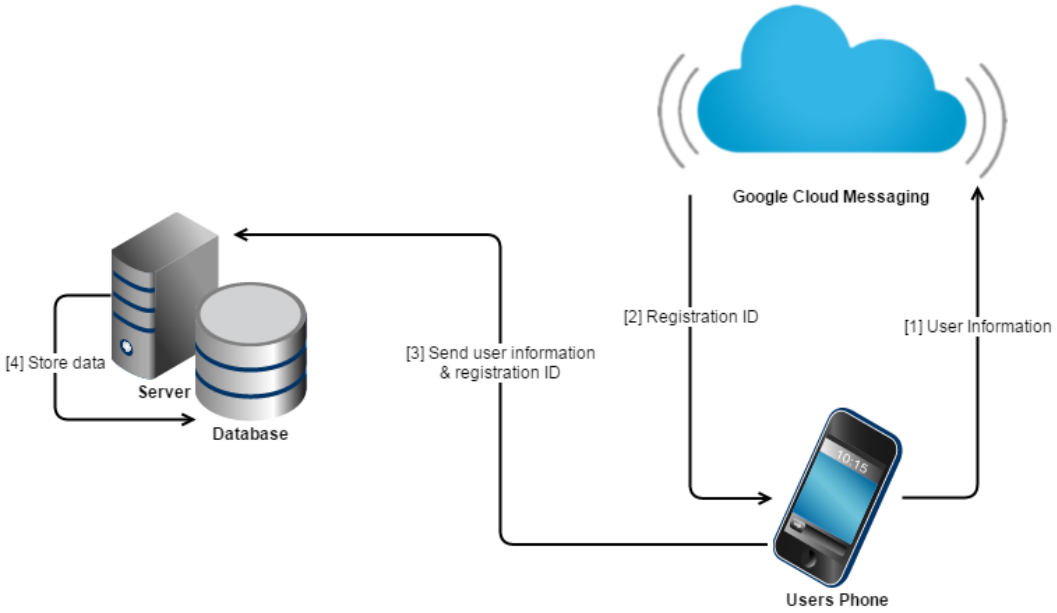

Figure 2: Diagram of the initial setup and registration with Google Cloud Messaging

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.

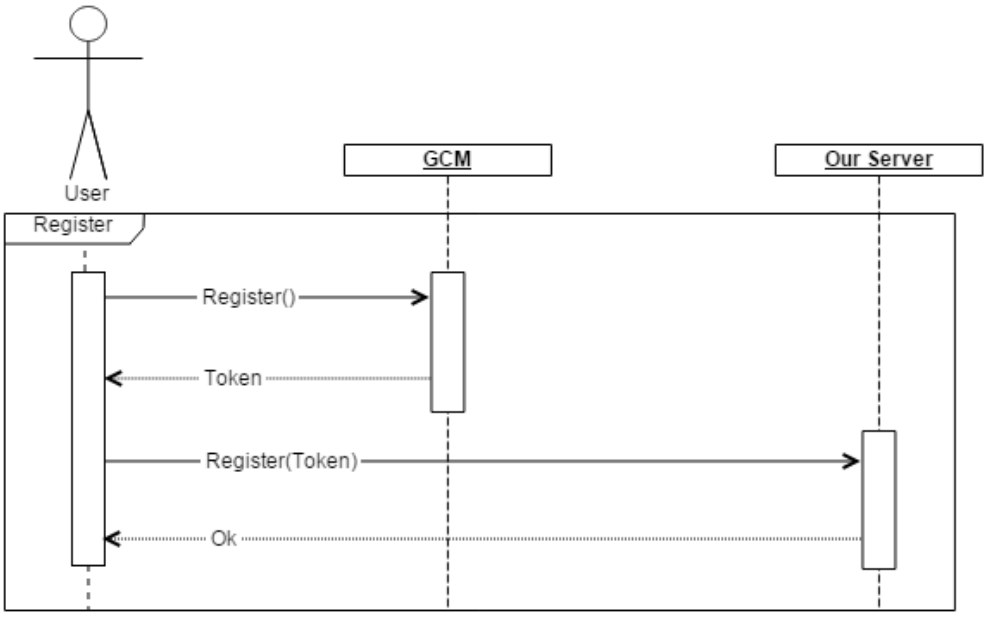

Figure 3: Sequence diagram showing interactions when registering a device

6.1.3 Sending Downstream Messages

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

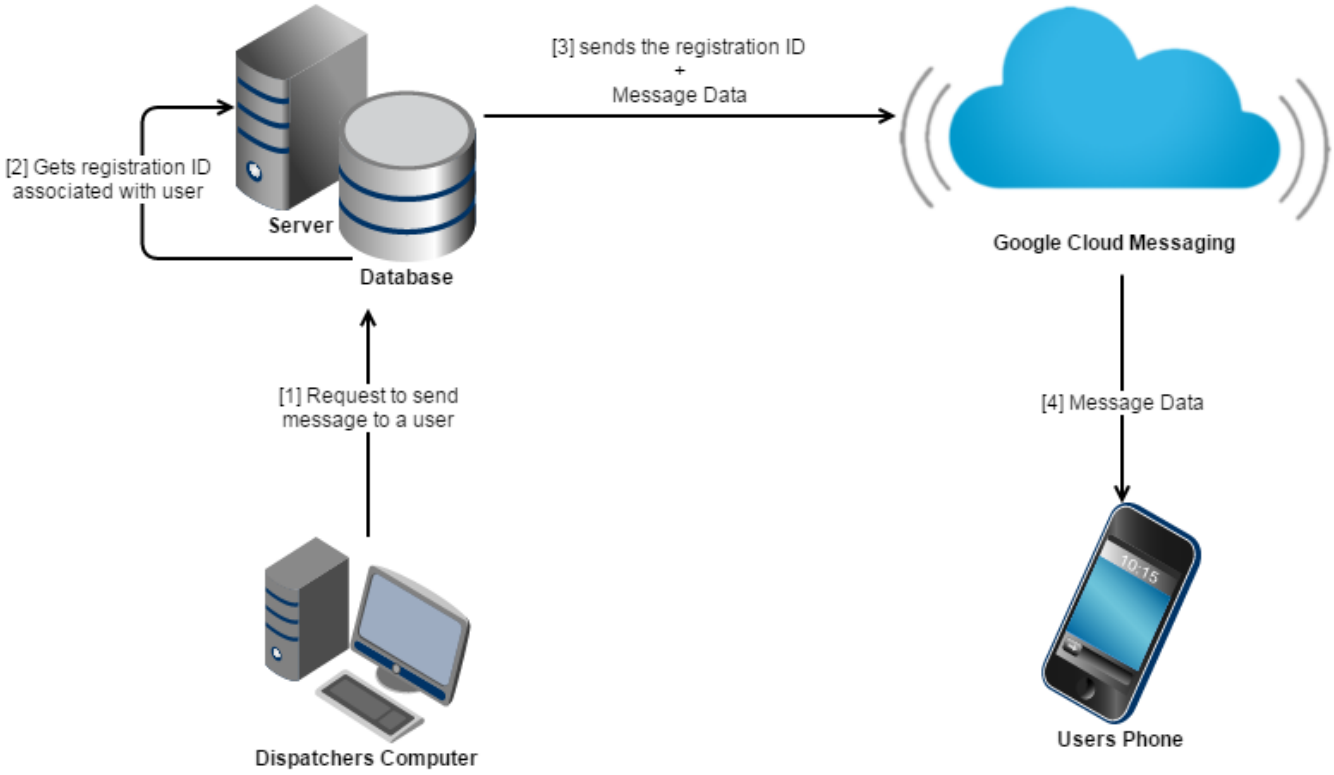

Figure 4: Diagram illustrating sending of a message to a clients device

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

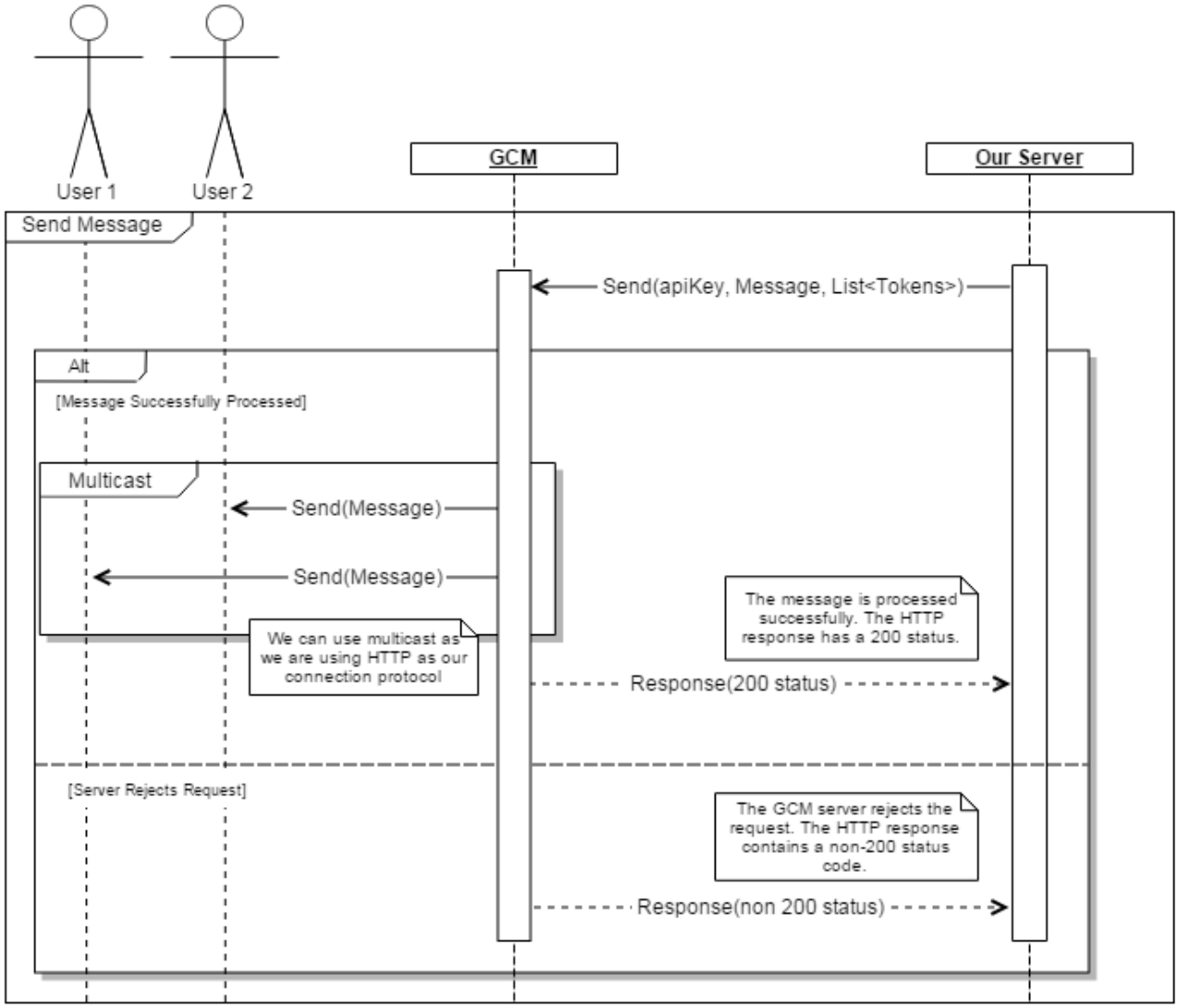

Figure 5: Sequence diagram showing the interactions when sending a message to the client device

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].

6.1.4 Sending Upstream Messages

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

Figure 6: 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>
```

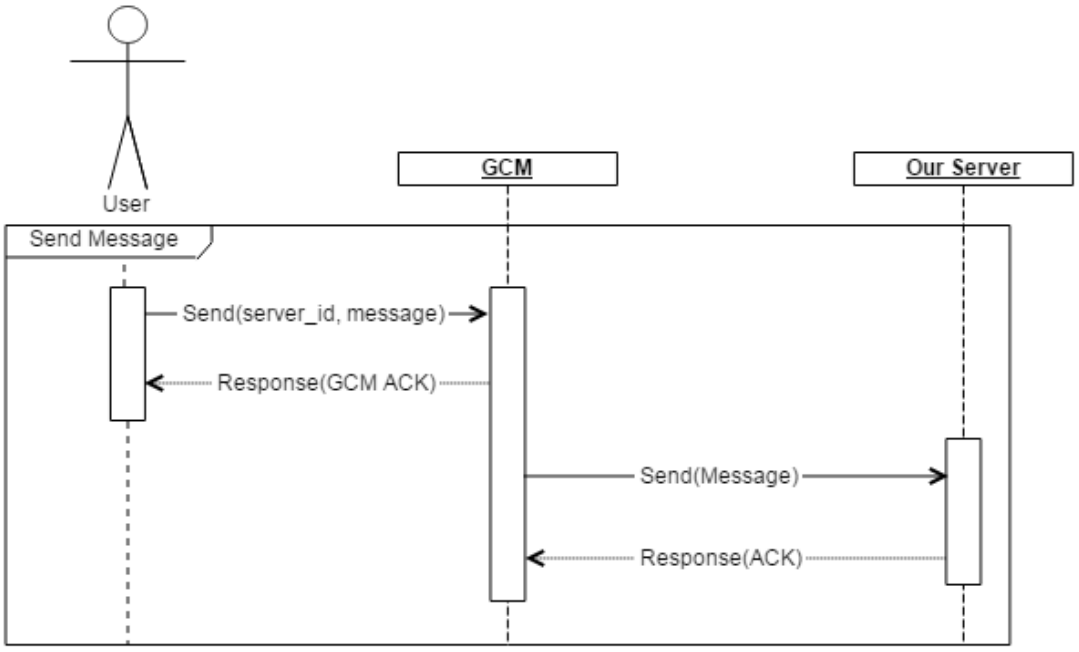

Figure 7: Sequence diagram showing the interactions when sending a message to the server from a client device

7 Project Management

8 Personal Reflections