Internship Report
Phasorz Technologies

Surajkumar .H., EE11B075

August 25, 2013

Abstract

A short report on the work done by Surajkumar .H. at Phasorz Technologies for the period of May-July 2013. The company is a product start-up currently focusing on their mobile-ECG product Dhilcare. Worked on multiple projects and modules which go into the final product + application whilst there.

Contents

1 Project Outline

1.1 Company and Product Background

Phasorz technologies is a product start-up founded in 2011. It aims at developing affordable cutting-edge products to suit developing markets needs. Phasorz technologies has a unique synergy of electronics and computing to deliver quality products. They shifted from a consultancy based company, outsourcing projects from Japan, to a pure product based company. Their major product in-the-works, is Dhilcare, the mobile-ECG device.

Dhilcare is an attempt to move medical imaging data away from the traditional ECG machines found in hospitals, to a tiny devices capable of receiving signals via sensors. Information is captured from these, robustly filtered, and sent via-bluetooth to a nearby Android phone. The product relies on the large-scale migration of users to smartphones, and this product was developed with this idea in mind. The phone receives the incoming Medical ECG data, and plots it, using the the smartphone's processing power to remove the reliance on traditional ECG Machines.

The data captured on the phone can now be transmitted to doctors anywhere. Since smartphones have inbuilt WiFi and Data modules, and because they are gaining such popularity among the users in developing nations too, it provides an ideal platform for sending data. Medical imaging data can now be sent to a doctor who can be many hundreds of kilometres away. This provides the ability for long-distance consulting of doctors using smartphones.

To better leverage the power of this idea, we also set about developing an application, running on the Android platform, to better suit our custom needs. Using this, doctors can capture ECG data, either using the hardware product, or using the on-board Camera / SD Card, and transmit the image to a specialist. The imaging data is stored on our own server, and sent to the doctor. The 2 doctors can now converse via messages, and come to an accepted diagnosis. The doctors can also create their own annotations on the image using our application, and share any relevant information with the patient.

1.2 Project Description

I worked primarily as an R&D for the application, a web developer, system administrator, and many other short roles. Whilst there, I was responsible for creating APIs for the mobile application to use, maintain servers and version controlled code, and also more involved tasks like encryption, data security, audit logging, crash reporting, medical standards compliance, analytics, data fragmentation and so on. The role involved learning a lot of new technologies, whether or not to integrate them in the still-beta product, and finding new/ modifying existing solutions to work for our requirements. Here, the technologies used will be discussed in detail, attempting to not reveal too much confidential information about the application and the exact working and ideation.

The major part of development happened in PHP and Android. The server used by us was an Amazon EC2 instance which will be explained in detail later. version controlled code was maintained in Bitbucket. The major interaction beteen the application and the server happened via POST requests to a particular URL. Since the scripts were now fully used for any web interface, they become APIs. The data was stored through a combination of MySQL databases, pseudo-randomly named images, both on the server and on Amazon's cloud S3 Storage service, accessible by API calls.

I worked on a number of small and independent modules that all came together to create a market ready prototype. One such was Encryption in storage and transmission of medical data, which is required as per HIPAA compliance. This involved setting up robust backup mechanism, making use of SSL certificates on the server and Android using public and private key encryption. I also worked with image compression technology analysis, data fragmentation for distributed transmission over Bad internet connectivity, protection against MySQL injection, setting up a basic web application for patients, creating simulations of live streaming ECG data on the phone and server side as a proof of concept, setting up crash report system on the Android side, and much more.

2 Humble Beginnings

2.1 Server Setup

As mentioned earlier, the servers used for the application are on Amazon's EC2. Amazon offers highly reliable and scalable server space, database management, and cloud storage. It is ideal for both the start up, and for the huge corporation to manage their information. They use a cluster of servers together, and create virtual partitions, allocating one to each requesting user, with full root access over this virtual server.

The Virtual Server space is called an instance, and for all purposes can be thought of as a black boxed PC running the server OS of your choice. We chose Ubuntu 12.04.2 LTS for the server space, for easiest transition for later employees to use, as Ubuntu is the simplest of Linux distros. Many big names do use this, including Quora, Coursera and so on, as scalable computational power from as low as running a web service, to cracking problems requiring supercomputer's processing power can now be rented

Setting up an Elastic Cloud Computing instance requires only a few button clicks, and some basic setup. Accessing this is done using Terminal / Cygwin or any SSH client of choice. One thing to be noted is that the mean of authentication is SSH keys. This removes the need for a password, while maintaining brilliant levels of security over access.

Amazon EC2 instances offer highly robust means of control too. You can block the HTTP / HTTPS ports, create security groups for access, block IPs from certain geographic locations, and much more from their side. All your information is backed up over several redundant images, and it offers the ability to take Hard Drive snapshots for system backup, Dynamic IP allocation for Web Hosting, and much more.

2.2 Version Control

We maintained two different servers, one for development, and one for deployment. The code was between the 2, and we used Bitbucket for version controlling. Bitbucket uses Git for version control, but provides private repositories, ideal for company projects. They make revenue off large projects, and charge based on number of collaborators. Project creators can set privileges for users, and in all other senses, it is perfect for maintaining version control, complete with branches, commit history, and the works.

The setup thus, was a Linux Server, Apache HTTP Client, MySQL database and PHP for coding. We also used phpMyadmin for database management, and GNUPlot for rendering ECG images. The standard development environment was an alternation between vim, emacs, and miscellaneous text editors rising and falling in popularity over weeks. The Android code was on Android Studio, and we used several phones for testing. Over time, this setup evolved in many ways, to suit security requirements, multiple session access, auto-timeouts, data backup systems and more, which will be discussed soon

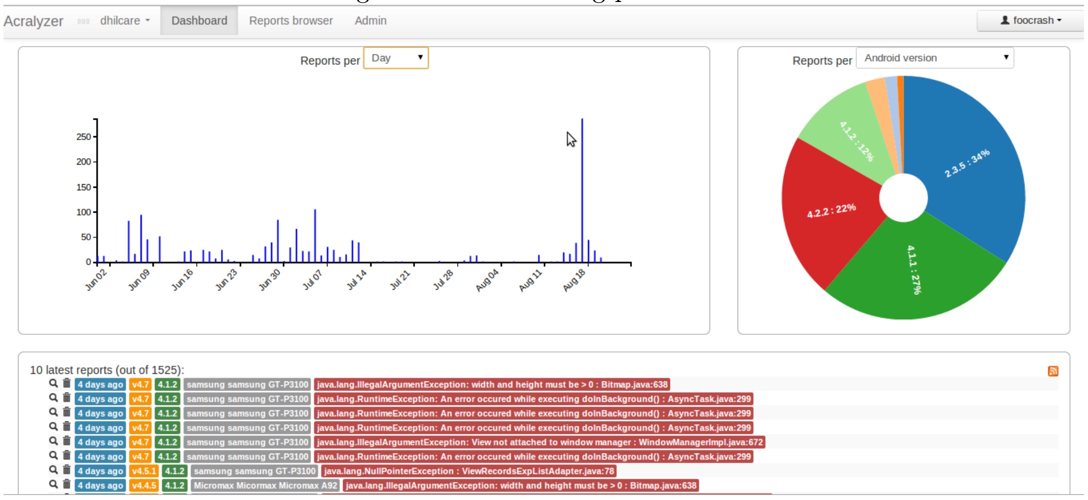
3 Crash Reporting

One of my first tasks was to get an up and running crash reporting system on Android. Android application crashes are quite common, and can happen for many reasons, from code issues, to RAM problems and or network connection problems. We needed a good way to monitor what problems the users faced while using the application. Since this was my first week, I was looking for a great contribution I could make to the project with existing libraries, which are aplenty, since I was still feeling my way around the existing code base, and the company's working.

Application Crash reporting for Android is an open source project for crash report capturing. It captures any and all exceptions on the Android side for crash reports, and also allows for custom information, which in our case was the doctor relevant information, phone number, ID, etc. to be tagged along and sent off to a NoSQL Database. This was then taken up by CouchDB, which offer scalable NoSQL database systems in the cloud, and by IrisCouch, which allows for control and regulation of this in an easy fashion. We also had a front-end UI for displaying crash reports, and statistics. A lot of this was plug-and-play, and required only minimal tweaking, but this system was quite invaluable as time went on.

This created a framework by which we would have a constant passive feedback from our application, the user diversity, the nature of crashes, whether they were version specific, and a good idea of what actually went wrong, and enabled us to make a great number of changes and tweaks during prototyping.

Figure 1: AcrA viewing panel Screenshot



10 latest reports (out of 1525):

4 days ago v4.7 4.1.2 samsung samsung GT-P3100 java.lang.IllegalArgumentException: width and height must be > 0 : Bitmap.java:638
4 days ago v4.7 4.1.2 samsung samsung GT-P3100 java.lang.RuntimeException: An error occured while executing doInBackground() : AsyncTask.java:299
4 days ago v4.7 4.1.2 samsung samsung GT-P3100 java.lang.RuntimeException: An error occured while executing doInBackground() : AsyncTask.java:299
4 days ago v4.7 4.1.2 samsung samsung GT-P3100 java.lang.RuntimeException: An error occured while executing doInBackground() : AsyncTask.java:299
4 days ago v4.7 4.1.2 samsung samsung GT-P3100 java.lang.IllegalArgumentException: View not attached to window manager : WindowManagerImpl.java:672
4 days ago v4.7 4.1.2 samsung samsung GT-P3100 java.lang.RuntimeException: An error occured while executing doInBackground() : AsyncTask.java:299
4 days ago v4.5.1 4.1.2 samsung samsung GT-P3100 java.lang.NullPointerException : ViewRecordsExpListAdapter.java:78
4 days ago v4.4.5 4.1.2 Micromax Micormax Micromax A92 java.lang.IllegalArgumentException: width and height must be > 0 : Bitmap.java:638

4 GCM and the war on Polling

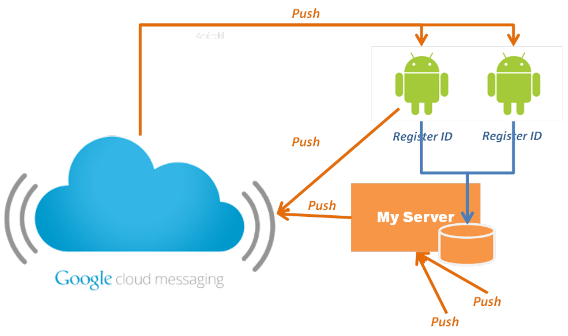
The next item of research was Polling vs GCM. We require information to be continuously synced between the doctor's devices. Every time the doctor clicks an image, or sends a message to consult another doctor, it goes through our servers to reach him. However the 2nd doctor has no information on when the other doctor has pinged him, what the nature of the request was, and so on. Naturally, we need a foolproof and immediate system for transferring information from one place to another. The 2 things that came to mind, were polling and interrupt services.

4.1 The basics of Pollin and GCM

Polling is where the devices continuously ask for updates or changes, based on past information. If any new information has arrived, it is sent immediately received, and the cycle repeats. This system is unbreakable, as it just keeps asking for updates, but requires a lot of memory and data consumption. Earlier, all devices including keyboards used to work off polling services, until such time interrupts came along.

Interrupts detect when change sin system state occur, and cause appropriate events to be carried out subsequently. Microprocessor interrupts are well known for this, and they are used for resets, and a number of other things. Interrupts are much more robust, and consume much less of everything. The only very large scale interrupt service on Android is Google Cloud Messaging or GCM

Figure 2: Overview of GCM



GCM works primarily by pushing data, and interrupts from a black box perspective. Devices can register with GCM, get a long but unique key much like a phone number or a primary key, and send messages to our server. Our Amazon server then receives this, logs the message request, and then sends it off to GCM. Google's server then receives our request, and forwards it along to the appropriate user. In theory, it sounds very simple, but applying with near 99 % reliability proved to be quite the challenge.

4.2 Troubles and Solutions

The issue here is that Google tracks the devices using heartbeat messages. Once every half-hour or so, the application sends a message to Google, informing it of it's exact location using it's IP address. Google then notes this down, and knows where to forward the message when it comes. But when this process fails, say when internet is off for an extended period of time, or at exact flashes of time, the messages fall through and don't get sent. Ultimately we did manage to et around this issue, but we were of the opinion that GCM wouldn't be a completely reliable system.

Temporarily, there was a switch to polling. This was a tremendous data churner, as there were 7 services, each logging in independently and maintaining polling services at ~10 sec intervals, which can cause server slowdowns / network traffic issues. The first solution here is to reduce Session timeouts and Cookie lifetimes, forcing the services to log in repeatedly rather than the default 2 hour auto log in, and thus freeing up valuable server memory. Then, a time-weighted polling service came into effect, with large bursts while using the application, and much lower frequency during inactivity.

A hybrid of Polling and GCM was eventually used, where one detects where the other is live, and appropriately scales up or down the polling service times. This sought to minimize the usage of polling as a technique, while keeping reliability levels high.

5 SQL injection

The existing codebase handed over had quite a few loopholes in security, and design. I sought a reason to rewrite a major part of the code, make it more modular, secure, understandable, and document-able. This appeared in the form of SQL injection.

The major problem here was that all inputs came directly from the user, and go through no validation process. The user input is plugged into an SQL query, and directly used to obtain results, without any validity checks. This is where SQL injection comes in. When certain types of inputs are not escaped, and several uncommon sequences not escaped, the user can enter certain combinations of input that will bring the system to it's knees

Consider, for example

```
“ SELECT * FROM doctors WHERE username = “ + $user_input + “;”
```

This is a fairly standard query used for login processes, where the user enters his username and can get logged in / relevant information sent to him. Consider the user input as,

```
’ or ’1’=’1’ /* ’
```

This effectively comments out the rest of the query, and since 1=1, any user can use this input to gain access to confidential information. Or worse yet,

```
SELECT * FROM users WHERE name = ’a’;DROP TABLE users;
 SELECT * FROM userinfo WHERE ’t’ = ’t’;
```

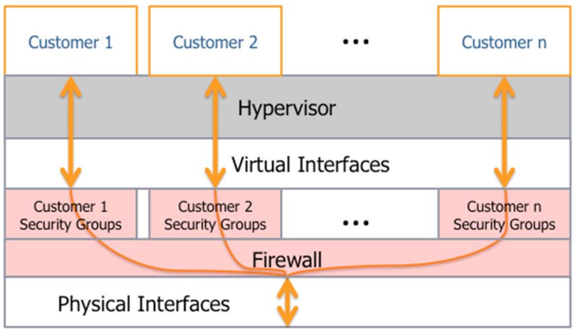
Where with a well placed user input, he can drop the entire table, leaving the application and product hanging on a wire. The major problem here is that most implementations of PHP libraries, even the traditional ones face risk of injection. One common solution is to strip all escape characters, quotes, and semi-colons from input, but even this can be circumvented.

The entire code was rewritten in PHP Data Objects (PDO) style, so as to improve code robustness, capture any and all errors and exceptions, and more. This captures all input variables, removing all traces of issue, binds them to a prepared statement. Then only it is executed, with cautions and error checks run quite often. This was also a great opportunity to comment every single idea / concept and deprecate redundant code, recycle and use global functions..

6 HIPAA compliance

A major part of the project was HIPAA compliance. HIPAA is the Health-care information Portability and Accountability Act, which defines a set of rules to be followed by application transmitting Electronic Protected Health-care Information (ePHI) over the internet. This was later followed by the HiTech rule which further added clauses for compliance. There are several major points to be covered to be HIPAA compliant. Amazon takes care of a few of them. Amazon EC2, and Amazon S3 (Simple Storage Service) provide simple, scalable solutions which are instantly HIPAA compliant, as they have redundancies over several physical locations, off-site backups, port restriction, ability to block access, Amazon employees not having any control / influence over the private server, and full control to the server's operating system.

Figure 3: Amazon and implicit HIPAA compliance



The major aspects not covered by Amazon in HIPAA compliance are Encryption and Security, Audit logging, and Database Backups. Database backups are provided by Amazon if going for their Relational Database Service, but a self-implemented solution is not difficult.

Code Snippet : Given below is an excerpt of a query used in the application, run in PDO style of database querying in PHP, protected against SQL injection, and with exceptions and sanity checks at regular intervals. The data is returned as a JSON object, and the motivation is to collapse conversations and consultancies from multiple doctors on a single record, into a single output message per record, and list the ones in the conversation, like a typical message / inbox style display.

```
function viewListOfRecentMessages($phone_number,$doctor_id) {
    try{
        $link=linkToDhilcareDB();
        $handle = $link->prepare("SELECT *  FROM
                     ( SELECT a.id,a.from_doctor_id,a.message,a.to_doctor_phone_number,
                       a.ecg_data_id, a.is_received,a.is_read,
                      a.creation_date FROM messages a
                     WHERE a.from_doctor_id=:doctor_id OR
                        a.to_doctor_phone_number=:phone_number
                     ORDER BY creation_date DESC ) c
                      GROUP BY c.ecg_data_id  ORDER BY c.creation_date DESC ");
        $handle->bindParam(':doctor_id',$doctor_id);
        $handle->bindParam(':phone_number',$phone_number);
        $handle->execute();
        $outer=array();
        while($row=$handle->fetch(\PDO::FETCH_OBJ)){
        try {
            $inner=$row;
            $ecg_id=$inner->ecg_data_id;
            $handle_names = $link->prepare("SELECT a.name FROM doctors a,
messages b WHERE b.ecg_data_id='$ecg_id' AND
                                   ( b.from_doctor_id=a.id OR
                                   b.to_doctor_phone_number=a.phone_number)
                             AND a.id<>'$doctor_id' GROUP by a.id
                             ORDER BY b.id DESC LIMIT 3");
            $handle_names->execute();
            $row_names = $handle_names->fetch(\PDO::FETCH_OBJ);
            $inner->name1=$row_names->name;
            $row_names = $handle_names->fetch(\PDO::FETCH_OBJ);
            $inner->name2=$row_names->name;
            $row_names = $handle_names->fetch(\PDO::FETCH_OBJ);
            $inner->name3=$row_names->name;
            array_push($outer,json_encode($inner));
        }
            catch(Exception $a) {
                $mess=$a->getMessage();
                return "[{\"success\":0,\"message\":\"Unknown PDO Error in
                       View list of recent messages, name finder\",
                       \"id\":$row->id,\"logcat\":\"".$mess."\"}]";
            }

        }
        return json_encode($outer);
    }
    catch(Exception $e) {
        $mess=$e->getMessage();
        return "[{\"success\":0,\"message\":\"Unknown PDO Error in View
                   list of recent messages, overall\",\"logcat\":\"".$mess."\"}]";
    }
}
```

6.1 Encryption and Security

Encryption is necessary for secure transmission of information. It is very simple for anyone to snoop in on network traffic, capture packets, find out the website it is headed to, and possibly even modify them. This turns into disaster if confidential information like passwords, banking credentials, etc. are transmitted without encryption as plain text. Anyone can find out dangerous information about you just from a network traffic monitor. So, we have encryption. No banking website is allowed to make online transactions without having a secure encryption scheme. A similar rule holds here in HIPAA, with medical imaging compliance.

In order to do this, a public-private key based encryption system was used, one for encrypting data and one for decrypting. The public key can be transmitted over not secure channels, and still be fine, as it requires a private key too. Usually, these 2 are linked by a mathematical relationship, like integer factorization problems which are harder to pull. Information can now be transmitted and received without much fear. But even this system can be broken in time. Hence, servers use certificate to communicate.

An SSL certificate is a verification of a trusted user in some sense behind the server, and to prepare for encrypted transmission. The SSL certificate on both the client and the server must be valid, else errors will prop up on instantiating. The server and the client check the validity of the other's certificate, perform a handshake, and spontaneously generate a private-public key pair, and exchange public keys. This process ensures that every session with a user uses on a new key, and hence is relatively infallible.

Implementing the SSL certificate on the website and Android was one of the tasks I was given. In order to do this, I had to use the openSSL module to generate key pairs and certificates. I created a self-signed certificate, which is bound to change long term, but is a great prototyping stand-in solution. After signing the certificate and changing the Apache configuration to listen on the HTTPS port and opening said port up in Amazon, created a rudimentary App for Android to be able to read the servers' certificate and continue on. Since the certificate had no trusted signatory, an Accept all certificate system on Android was used as part of a test application. Self-signed certificate usage in Android using BouncyCastle was considered, but after the set validity period, when the certificate expires, the application becomes unusable as the server pings over HTTPS fail due to no mutually recognizable certificate. An accept all certificates system worked well for the beta application, and a move to a trusted signatory is considered to be best during deployment

6.2 Audit Logging

One of HIPAA's major compliance requirements was a comprehensive log of any and all modifications / interactions by users on the application. This allows for strict user control / monitoring, on the database side and login side of the application. This allows the system administrator of any users with malicious bots who modify data indiscriminately, allowing for their IPs to be blocked. Many existing Audit Loggers did not have quite the functionality required, so an independent one was written.

The Logging happened in the API calls themselves. This assumes that the server itself is largely untouchable, and that any vulnerability needs to go through the API calls, the chink in the armor. This is a valid assumption, given how we tightened down on security, password protected portions of the website, changed user privileges, and did frequent password shuffles. So, all Logging happened in the API calls, where a separate database of logged information was maintained, also in SQL. Whenever an API is called, one which is either a login script, or an attempt to modify information, or upload / download data, it is logge into one of several tables, with the nature of information requested, user ID, user's IP address for possible banning, Session data token which can quickly remove his existing information, and may require a change in strategy, and levels of accepted input, whether the login / database modification is a routine application request or a unknown attempt to hack the server.

6.3 Database Backups

Database backups were another major requirement of HIPAA. The application or website shouldn't have to call up it's users and tell them their data is lost because the server crashed. Data redundancy goes a long way towards assuring the user his information is safe. A robust database backup mechanism was required.

Automated Version controlled database management seemed like a great way to go in this regard. Since every commit / version is likely to contain lots of new information, it doesn't make sense to store one for each date-time backup, rather storing just the difference should suffice. Here too, BitBucket was used, as it enables us to reconstruct commits and recover data from any instant in time, while ensuring that sizes were kept small, as only differences in data were stored each time.

This commit push process had to run continuously, at admin specified intervals of time. So, a cron job running a git-based push script running over SSH keys was used, which grabbed the database, and sent it off to the BitBucket repository, storing the difference, time of update and so on. SSH keys were needed to remove password reliance, and ensure that even automated bots and scripts can do Git interactions. This method of management worked out well in the short run, though better database systems like the RDS systme were considered for use after deployment.

7 Image Compression Formats and Data Fragmentation

One other major consideration while making the application was data usage. The application would be used in part by doctors and nurses, who may be from remote villages, or hospitals with great facilities, or even a moving ambulance. The application needs to be robust enough to be able to send data even when on the move. This wasn't really a problem when it came to the messages, since they involved a simple POST request, which can either succeed or fail, in which case resends are attempted, or waiting for a change in network state. The real issue is when it comes to images, camera images and the 12-lead ECG from the device. We need ways to send this, and still bear with drops on network connectivity, and slow internet speeds. This was more like a research problem, where hours were spent on white papers, comparisons, alternative products and ideas, and so on.

7.1 A comparative analysis on Image compression formats

There are various openly available image compression formats. These include, the Raw image, the TIF, the BitMap, lossless compression formats like GIF and PNG, and lossy compression schemes like JPEG and the upcoming JPEGmini standard. These operate in very different ways.

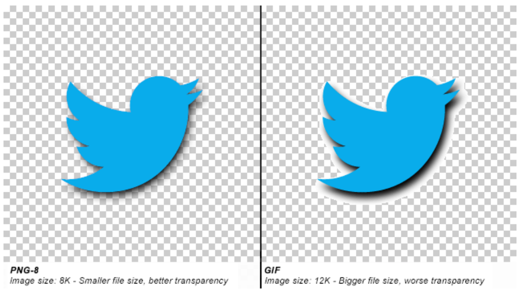
The lossless compression methods create a dictionary of sorts, or a reference between similar colors. It picks out one pixel which is quite commonly found in the image, and makes other pixels with nearly the same value refer to this pixel. Since no information is actually thrown out, this is pure lossless compression, very similar to the zipping idea of compressing files based on dictionaries of similar data. It is possible to get very close to the original image on reversing the compression process. Lossless compression methods include GIF and PNG.

macy's
GIF
Image size: 4K - Better quality, smaller file size
macy's
JPEG
Image size: 11K - Worse quality, larger file size

Lossy compression on the other hand actually throws out many pixels which have little or no differential contribution. This involves significant loss of information, but also makes for great versatility depending on the situation. The JPEG image format is the most well known here. It involves taking a Direct Cosine Transform of the original image, finding a frequency domain representation of the image, quantizing and filtering out some high-frequency signals, sequenced and then transformed back. Since this acts like a low-pass filter of sorts, it does involve significant loss of information, but also gives the user the ability to define the compression ratio, and what percentage of the pure image quality is to be retained.

JPEG
Image size: 38K - Better quality, smaller file size
GIF
Image size: 80K - Worse quality, bigger file size

The need for the application differed in different contexts. The images captured by the phone camera were usually of a very high resolution, given that all testing cameras were above 5 MegaPixels in resolution, which translates roughly to about 750 KiloBytes per JPEG Image and 1.8 MegaBytes for GIFs, and 3.8 MB for PNG. In any case, such a high resolution, a.k.a 5 million+ pixels may not be completely necessary for viewing on mobile. Some services get away with compression levels up to 70 kB per images. So we experimented and found that without much loss in data, we could hit the 100-200 kB limit while transmitting images.


PNG-8
Image size: 8K - Smaller file size, better transparency
GIF
Image size: 12K - Bigger file size, worse transparency

Another factor was differentiating between the various file formats. JPEG is considered much better for camera snapped images which have very high color variation, and rich images in general. For images with more line drawings, and less color, Lossless compression works out better, PNG can outperform GIF by upto 15% compression in these scenarios, better helped by the fact it has alpha channel for transparency.

The net result was that images were scaled down to very low resolution, and JPEG was used as a compression format for camera images, as the diversity of camera inputs is unpredictable. A network state based predictive scaling was used, where the down-scaling factor increased as the network connectivity / speeds were lower. This enabled people who had the means of sending heavy files were able to fully well do so, while the emergencies could warrant a quick switch to the high compression state.

7.2 Data Fragmentation for reliable uploads

Once the issue of minimizing camera captured input images was handled, the next task was to understand the process of transmission, and how to possibly take charge here. The service needs to be up and running even if the doctor needs to consult in the frenzy of the moving ambulance, jumping between base stations and losing connectivity often. This requires fragments to be sent whenever they can, in whatever size possible to reach, and without having to send again within a certain time interval. That is, if fragment X was sent now, X shouldn't have to be resent if the connection fails and is restored within one minute. Unlike any casual applications where loss of information is tolerated, here it could be mission critical to get the information across as soon as possible

The inspiration for our idea came from an understanding of how HTTP-like protocols actually deal with data. All data is split up into chunks, and chunks are sent serially (possibly in parallel too). Each chunk has a unique identifier, and chunks are sent individual of each other. There is a certain time out for any HTTP request, and within that time, packets are sent captures, and pieced together at the server end. There is no real requirement for the packet id based sorting in serial transmission, but it provides a sanity check, and proves necessary when it comes to parallel channels for transfer.

We also drew parallels from MIT's Sana open-source project, providing free and open healthcare to third-world countries, and where doctors used to consult, send back questionnaire responses and images over phone to researchers / doctors in the US, who would volunteer to help and diagnose.

In third-world countries, guaranteed high speed internet is practically non-existent, and hence they came up with an ingenious method to transmit information, which we modified for our application

All images / heavy information (which falls beyond a simple POST / URL request) is first converted into an equivalent binary file, with the headers sent first to the server. The device would then send an initial random-but-small amount of data, say 2 KB. All uploads happen serially, and hence the server sends a custom ACK (Acknowledgement message) after receiving and storing the packet. This information is stored in a temporary binary file on the server. The phone then tries to upload the next sequence. If the upload fails, it is possible that the size is too large for transmitting at this bandwidth, and hence we reduce the image size to 1 kB. If the upload succeeds, the upload is likely to be fast, and hence the chunk size doubles to 4 kB. If the connection is stale for more than, say 4 minutes or so, it is cut-off and the binary file is auto-erased.

This system provides it a feedback loop like mechanism, where ultimately, the chunk size stabilizes at some particular value and the image is sent. The server keeps adding binary chunks to it's temporary file. At the end of the process, the device sends a 'finished' acknowledgement, at which time the server re-encodes it into a complete file and moves it to the appropriate location. This is a very robust solution, and can be implemented for any nature of data, including video should it come along.

8 Smaller Projects

There were a number of smaller projects I worked on, for a few days at a time, which were rapid prototyping periods, proof of concept builds, and usually had steep learning curves and short deadlines. These were some fun projects to work on, and some eventually did queue up for integration into the integration, with all of them waiting in line for the same.

8.1 SMS Validation

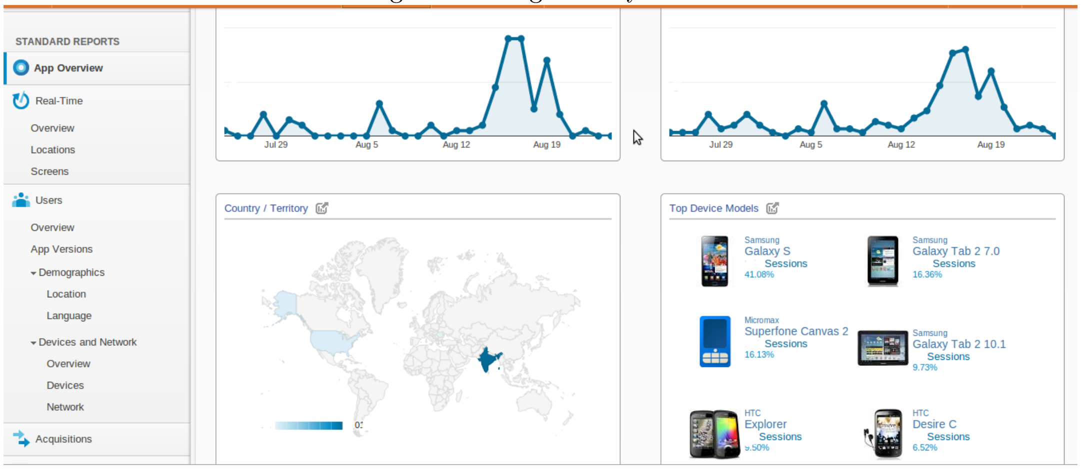
To make an application professional, it must provide fail safe options and recovery options, even cases of phones getting stolen, change in SIM, etc. All the data must be carried over to the new phone, and also the user must have the option to remotely lock his account.

In order to achieve the former, an SMS based validation system seemed most appropriate. Many big names which work off phone numbers use SMS based validation, as it is relatively more difficult to spoof. Two options were considered, one implemented, and one still being tested to achieve this. The options were Send and receive based Validation.

SMS validation by sending (from the server) is where the user enters his phone number, and our server generates a random number sequence and sends a text message to his phone number with this information. The application then detects the message / scans for the verification code, validates it, and then changes user privileges. This system worked out quite well, and all SMS sends were based off API calls from the server, which were offered by several services like Twilio, mVaayoo, over a transactional route for DND numbers.

SMS receive validation works in the reverse sense. The application sends a text message on the users behalf to a predetermined phone number The phone number has a service running on top of it which receives text messages from all phones, validates the phone numbers, and communicates with the server directly for access grants. This method removes the users need to contact the server, and is reliant on his phone's ability to send text messages, and is quite ingenious in some regard.

Figure 4: Google Analytics Tracker



8.2 Database and Application Analytics

Analytics is very helpful for the app developers to know of information relating to trends in application usage without getting any confidential information about the user. This helps the producers tweak their market strategy according to user trends. Two kinds of analytics is required, one for user demographic, and general information, and one for application specific, highly targeted information.

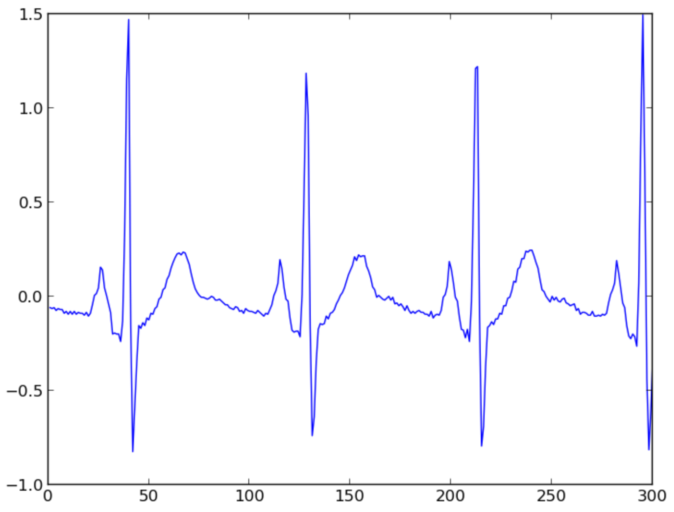
For the demographic based analytics, a hybrid of Google Analytics and Play Store Analytics was used. This operated largely by script calls from the Android side. Whenever a phone does some interaction, it is sent via-script call to Google's server, with loads of information about the location, phone number, engagement time, and much more. It provides a simple mechanism to track user trends in data.

For more custom application specific, data driven analytics, we needed a highly customizable service which provides ability to easily add and remove developer-specified traits to track. To do, this, Mixpanel was used as a solution. Mixpanel provided highly customizable, application specific analytics as a solution. It enabled us to set up user traits and trends to track, define events and activities. Whenever the user performed a traceable event, information is sent to Mixpanel's server via script call, and they stpre the information on their server. The provide solutions for easy access to complex queries on their server, and can deal with mounds of data, hence were chosen for analytics.

8.3 Auto-updating of application

Auto updating of application involved setting up a mechanism to release app version on the website, track versions maintained by each user, and notify him whenever a newer version is released. User app version is tracked as aprt of analytics, and the same information is sideloaded here. Application versions are uploaded onto the server, the changelog, version, and other details stored in an easy trackable file. Whenever the user logs in, the application pings the server for updates, which compares it's current version with the server's most recent one, and if it detects an update, sends all the relevant information to the user. This enables us to have stricter policing of the release version of the application, and can serve as a useful tool even after deployment.

Figure 5: GNUPlot of Apnea ECG data



8.4 Pseudo-Live-Streaming ECG Data

A rapid prototype for Pseudo-Live Streaming ECG data was required for a proof of concept, and was developed. To do this, information was pulled down from Physionet's massive online open database of health-care information. Apnea ECG information was chosen for test driving the idea, the information was obtained, padded and scaled to one hour period.

The information was stored in a text file, with provision for looping backward and forward. The idea here is that on the application side, the user can continue to receive live data, and can also scroll backwards in time to obtain history of data for a certain limit on time. This involved storing all information received in a text file, real-time updated, which was simulated by an infinite parser of the one hour ECG data file. Live time-stamped information was sent to phones, where it was plotted, and if the user requests for ECG history, typically by backward scrolls on image, he can go back in time, while continuing to receive live information, and can scroll forward to current time whenever he so wishes.

This can be used for doctors to remotely monitor patient ECG information for nurses or on-duty doctors in the hospital to consult specialists far away, and send all pertinent information easily and rapidly, for them to monitor real-time patient progress, and provide accurate diagnoses.

9 Acknowledgements and Conclusion

For my experience over the summer, I would like to thank Enbasekar D, co-founder of Phasorz Technologies, and my project mentor, who helped shine light in the right direction, and could always help set priorities. I would thank Satish Kannan, also a co-founder, for his talks on how start-ups work, the market for healthcare, and the various stages in product development, from ideation to production to sale. I would thank my co-intern, Tejasvin, whom I collaborated with extensively on the application, and was a sounding board for both unique concepts and implementation details. I would thank my family for their support, during all the long hours spent working on perfecting tiny details.

The summer provided me with a great opportunity to understand start-up culture, experience and learning. As far as start-ups go, everybody is expected to know a bit about nearly everything going on, contribute ideas. You are expected to be the specialist whenever someone else is too busy to do so. It involves switching between rapid prototyping for instant deployment, to fine-tuning details which will behind the scenes improve performance, security and stability. A lot of ideas, white-board sketches, models and paper flow diagrams were were put forward. The start-up experience is one worth having, offering a micro-level understanding and an intense hands-on experience opens ones eyes to the amount of hard work that goes into every feature, that even the smallest details can make a difference, and that ideas require effort to see the light of day. The experience was invaluable, and the joy of seeing ideas in motion, all coming together to form a complete product from end-to-end is quite novel.