Ram Krishna Banstola

Implementing Push Notification Systems for Contextual Activity Sampling System

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Bachelor's Thesis

November 10, 2015

Author(s) Title Number of Pages Date	Ram Krishna Banstola Implementing Push Notification System for Contextual Activity Sampling System 35 pages + 7 appendices 10 November 2015
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Kari Salo (Principal Lecturer)

The primary goal of this thesis project was to implement notification systems for Contextual Activity Sampling System Query (CASS) developed at Helsinki Metropolia University of Applied Sciences. From an application point of view, a notification is event-based mechanisms which consist of a remote server that pushes events for a mobile or a browser client. This push event is triggered because new data is ready at server to be served to the client. In CASS, this new data is the new query that is ready for the research subject to be answered.

CASS is a research tool used by researchers to collect process and context sensitive data. It has a web tool where researchers can create new researches, download and visualize the data collected, a browser application which can be used by research subjects to answer queries and a mobile application for iOS and Android for answering the query from a smartphone. Currently this system is being tested by researchers at the behavioral Science Department of the University of Helsinki. The CASS mobile application lacked the push notification system which is an essential part of data collection for improving the quality of data collected. After the completion of the project the Android version of the mobile application will be equipped with push notification system using Urban Airship. Amazon's Simple Notification Service (SNS) was used to send email notification when a research query was ready to be answered, which was replaced using custom email notification service. PHPMailer, an open source PHP library, was utilized for sending the notification emails using Amazon Simple Email Service (SES) Simple Mail Transfer Protocol (SMTP) server.

Keywords	Push notification, Android, iOS, PHPMailer, CASS, Cross platform, SES, SMTP, WebSocket

Contents

1 Introduction 6

2 Theoretical Background of the Technologies Used 8

- 2.1 Email Notification 8
- 2.2 Text Message Notification 9
- 2.3 Mobile Push Notification Systems 11
  - 2.3.1 Google Cloud Messaging 11
  - 2.3.2 Apple Push Notification Service 13
  - 2.3.3 Windows Push Notification Service 14
- 2.4 Cross Platform Push Notification Systems 16
- 2.5 WebSocket Notification 18
- 2.6 Security Risks of Push Notification Services 20

3 Analysis, Development and Deployment 21

- 3.1 Necessity and Comparisons between Nnotification Systems 22
- 3.2 Implementation of Urban Airship Push Notification System 24
  - 3.2.1 GCM and Urban Airship Setup 24
  - 3.2.2 Urban Airship Backend Implementation 26
  - 3.2.3 Client-Side Implementation 28
- 3.3 Email Notification Implementation 30
  - 3.3.1 Amazon Simple Email Service 30
  - 3.3.2 Custom Email with PHPMailer Library 32

4 Results 34

5 Evaluation of the Results 38

- 5.1 Benefits 38
- 5.2 Challenges 39
- 5.3 Future Improvements 40

6 Conclusion 41

References 42

Appendix 1. Urban Airship API call from CASS server

Appendix 2. Sending email using PHPMailer and Amazon SMTP server

Appendix 3. Broadcast receiver class in CassHelper Application

Appendix 4. MyApplication class of CassHelper

Appendix 5. MainActivity class of CassHelper

Abbreviations and Terms

ADM	Amazon Device Messaging
API	Application Programming Interface
APNs	Apple Push Notification Service
GCM	Google Cloud Messaging
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IMAP	Internet Message Access Protocol
IP address	Internet Protocol address
ISP	Internet Service Provider
JSON	JavaScript Object Notation
NCP	Notification Client Platform
POP3	Post Office Protocol 3
SES	Simple Email Service
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
SNS	Simple Notification Service
TCP	Transfer Control Protocol
TLS	Transport Layer Security
UA	Urban Airship
URI	Uniform Resource Locator
WPNS	Windows Push Notification Service

1 Introduction

Contextual Activity Sampling System (CASS) is a web tool that facilitates researchers to gather data that are process and context-sensitive which are useful to study for example human behaviours and work experience in general. Based on the data collected during the research the researchers can analyse how human behaviour fluctuates during different times of the day, how different people experience different things at different times of the day etc. Thus a co-relation can be produced by using the data later on. CASS consists of an admin console for creating research, analysing data collected from the research, monitoring data, a browser application that consists of a simple user interface containing updated queries that research subjects have to answer and an Android application for the same purpose as the browser application. [1] CASS console is created using PHP, while browser application is created using AngularJS. This thesis, however, focuses mainly on creating notification systems that support both native and browser clients in an optimized so way that a subject can get a notification as soon as a query is ready to be answered.

There are four data collection methods in CASS namely fixed time, fixed interval, event contingent and randomized. A researcher can create a research for a period of time and queries are delivered every day based on the type of research method and frequency at which queries are to be delivered every day. In a fixed time method researchers assign a fixed time for answering a query every day, e.g. at two o'clock during day every day for five days. So the time is already known to the subject and does not require a notification in general. In a fixed interval method a time is set for answering the very first query and an interval in which to answer the rest of the query. In a fixed interval method, the next query is delivered to the subject based on the last time the query was answered and the time interval, e.g. if the first query can be answered after nine o’clock in morning and the time interval is one hour, if a subject answers a query at half past nine then the subject will get next query at half past ten. Similarly in an event contingent method a subject can answer a query whenever he/she wishes. The randomized data collection method is unique in the way that random times are generated by the CASS system. Researchers can create researches where query time is randomized every day or a random time is created once and it is used until the research is complete. Since a research subject does not know when to expect a query beforehand, a notification system is implemented for the randomized data collection method. The user's phone number or email address can be provided to the CASS system in order for the system to send an email and push notification. Giving a phone number to the CASS implies that emails will not be sent to the CASS research subject while giving an email implies the subject will receive both an email and push notification at their smartphone. Notification emails for randomized data collection are sent using Amazon's Simple Notification Service (SNS).Amazon SNS is a third-party service that is used to send emails and mobile push notifications using their API and infrastructure [2]. SNS is not reliable for CASS because of reliability issues such as emails not being delivered to clients or emails being delayed and difficult to debug to find the cause of email undelivered.

The project had two primary goals. The first goal was to implement a push notification system for an existing mobile client and the second was to replace the existing Amazon SNS email notification service with a custom email service to send notification email when a query will be ready to be answered as well as to inform user about subscription to research. Though new version of modern browsers supports notification using WebSocket, a protocol for full duplex communication between server and client over a socket. It is not supported by all browsers. Thus an email notification is used for notifying subjects answering the query using a browser client.

2 Theoretical Background of the Technologies Used

2.1 Email Notification

Email is an important collaboration tool for the successful operation of virtually any organization in the industrialized world. Email as a communication tool takes more time than instant messaging, social media and the telephone combined. What makes email use so pervasive is the ease of use to send and receive files and messages. So it is the primary information transport system for many enterprises. It is used to send automated messages such as transaction report after a user has bought an item over an online shop, for mass marketing by online stores in the form of newsletters, for sending monthly invoices by service providers to their clients etc.

A Simple Mail Transfer Protocol (SMTP) Server is necessary for sending an email while Post Office Protocol 3(POP3) or Internet Message Access Protocol (IMAP) is used for receiving an email. A program called email client, e.g. Microsoft Outlook is used for sending and receiving an email. [3] An email client is necessary for receiving emails but CASS only sends emails. So an email client is not necessary for CASS. Figure 1 shows the basic principal how an email works.

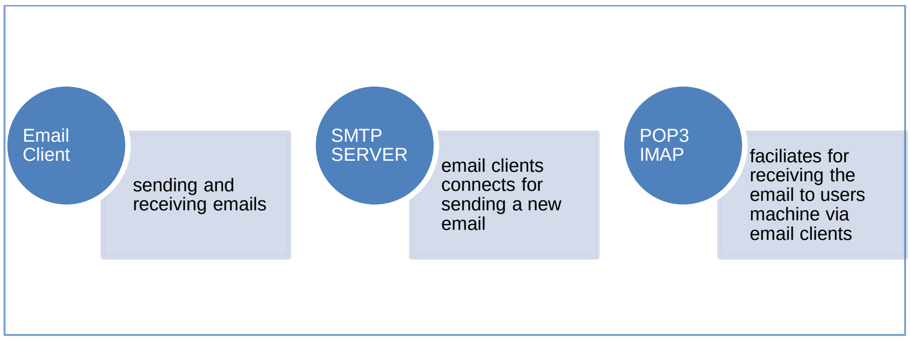

Email Client
sending and receiving emails
SMTP SERVER
email clients connects for sending a new email
POP3 IMAP
faciliates for receiving the email to users machine via email clients

Figure 1. A basic email sending and receiving process.

In the context of CASS when Amazon SNS was used two types of emails were sent to subjects who had taken part in research. As soon as a research admin entered email address of the subject, an email was sent to the subject notifying him/her of her participation in the research. This email contained a link from SNS informing user to subscribe to notification emails in the future. All other emails are sent exactly on the same date and time when a new query is ready to be answered. If the user failed to subscribe then no emails were sent to the subject in future whatsoever. This step is necessary to ensure that emails are sent to a valid subject. So it would be impossible to sending notification email if user failed this very first step. This helps to ensure the right person will receive the email. Figure 2 shows SNS email notification in CASS.

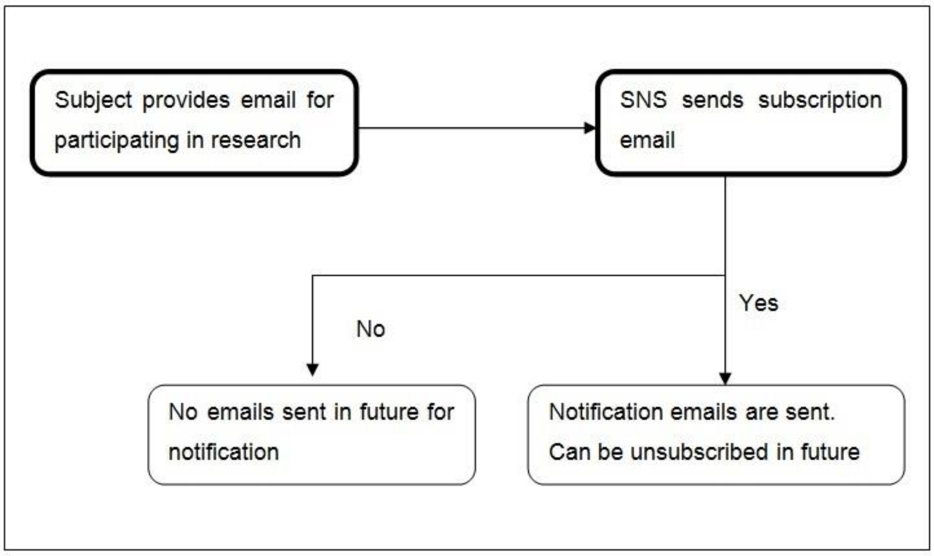

Subject provides email for participating in research
SNS sends subscription email
Yes
No
No emails sent in future for notification
Notification emails are sent. Can be unsubscribed in future

Figure 2. SNS email notification in CASS

Using a trusted vendor like Amazon is necessary for the success of the email system used by any organisation. Email system success is measured with four metrics namely bounce rate, complaint rate, content issues and delivery rate. Bounce is the state when an email cannot be sent to the intended destination address because the mailbox address does not exist or the mailbox is full. The complaint occurs when a recipient marks the email as spam or junk. A content issue arises when the content of the email is for example,malicious, misleading, scam etc. Delivery rate is the number of a successful email delivered to its recipient without having the issues like bounce, complaints and misleading contents. [4]

2.2 Text Message Notification

Text message is a widely used notification method by various applications and services. Most common purpose of an SMS notification in the present-day context of a smartphone application includes number verification, notification of certain events, mass advertisement, small payment method etc. Text notification is not very cost- friendly compared to email notifications even though it is a very effective way to notify a user about time-sensitive information. Text notification is popular among logistic businesses, education providers, health, banking etc. It is a reliable means of mass awareness in many developing countries as well as a means to rectify various services in developed countries. For example, during Ebola outbreak in 2014, Senegal sent four million SMS messages to general public in its attempt to raise public awareness about the disease [5]. Many emergency services are designed so that people who are in the nearby location where an emergency has occurred are sent an SMS about the incident and instructions to follow. In big enterprises such as DHL, SMS is a convenient way for users to track the progress of their order or receive information about where to collect their order from. SMS is handy when a mobile network is congested due to the presence of more than the expected number of devices in a network cell.

An SMS gateway is preferable in situations when a text messages is automatically generated based on the event. An SMS gateway is a web service, generally a website, which allows users to send an SMS using a computer [6]. Amazon SMS notifications are currently supported for phone numbers in the United States, thus instead of an SMS a push notification is sent to mobile users. Similarly sending notification email using SMTP, SMS gateway can be integrated in the existing CASS backend to implement sending of SMS.

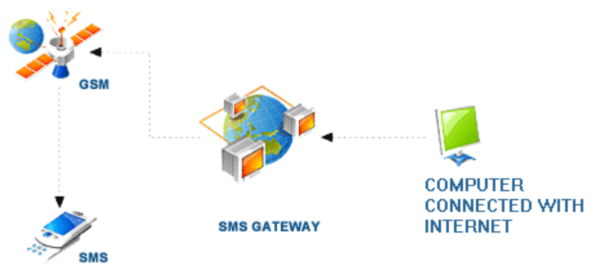


Figure 3. Use of SMS gateway to send SMS to client devices. Reprinted from Verma InfoTech [7]

As shown in figure 3 SMS gateway works as an interface between different protocols to send SMS created from a website or a desktop application. It converts and sends the messages to SMS Centre which is the location setup by Network providers to collect SMS and send it to the designated cell phone number.

2.3 Mobile Push Notification Systems

Smartphone applications have become an essential part of every day activities in the industrialized countries. They are used for easy access of services such as health, education, transport, navigation, shopping etc. Similarly they are also being used for events that are time sensitive for example notifying a car driver about an accident that occurred in his/her route and offering information about an alternative route or a delay due to the accident. Applications are so smart that they are communicating with the server almost in real time. [8] How do they do this? In a higher level abstraction, it is done by maintaining a constant connection between the application and the server. Mobile push notification systems are designed to fulfil this gap of maintaining this communication by application developers so that the application gets notified once new information is available in the server. Each platform has its own architecture for implementing a notification system even though the main goal is same.

2.3.1 Google Cloud Messaging

Google Cloud Messaging (GCM) for Android is a notification service that facilitates an application to send downstream messages that originate from the server and are destined for application as well as upstream messages that originate from a GCM-enabled application and are destined for the server. GCM handles all aspects of queuing of messages and delivery to the target Android application. There are two types of messages that can be sent or received using GCM, a lightweight message informing the application about new data that is available in the server, e.g. a new video has been uploaded. Another type of message can contain up to 4 kb payload data that enables applications like an instant messenger to send and receive payload. [9]

Figure 4 shows GCM communication on a GCM-enabled client application in the client application on an Android device. Figure 5 below shows the actual notification on an Android device.

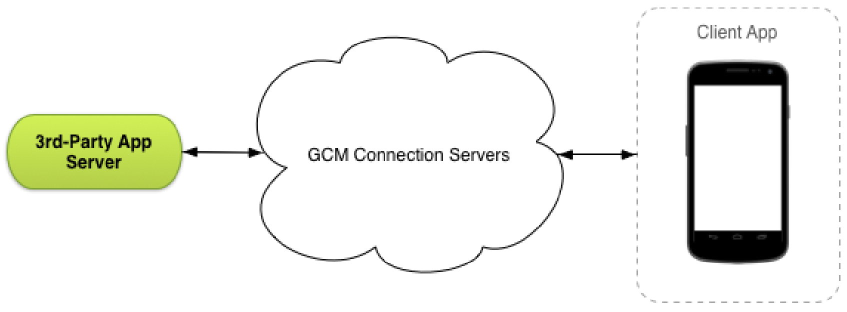
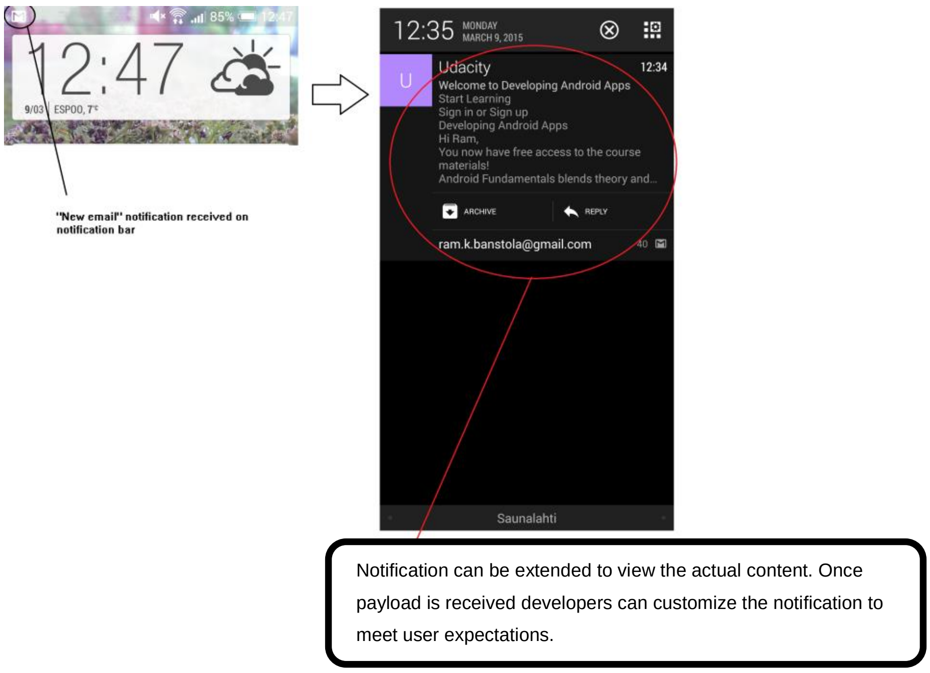


Figure 4. GCM architecture. Reprinted from Android Developer Documentation[9]


12:47
9/03 ESPOO, 7°c
"New email" notification received on notification bar
12:35 MONDAY MARCH 9, 2015
U
12:34
ARCHIVE
REPLY
Saunalahti

Udacity
Welcome to Developing Android Apps
Start Learning
Sign in or Sign up
Developing Android Apps
Hi Ram,
You now have free access to the course materials!
Android Fundamentals blends theory and...

ram.k.banstola@gmail.com

Notification can be extended to view the actual content. Once payload is received developers can customize the notification to meet user expectations.

Figure 5. Push notification on an Android device.

A GCM implementation consists of a Google-provided connection server, a third-party application server and the application itself. On the user device there is one active connection to Google's cloud server which is not power hungry as no traffic is sent through it until no data is available. All the GCM-enabled applications uses this connection for receiving data from their server. [9] This is a smart way to exchange data between applications and their server because it does not require a traditional polling action to check at regular interval for new data. Since all the applications use the same connection, it saves both power and data in a smartphone. Whenever any new data for any of the GCM-enabled applications is available, then this connection will be used. The application server sends data to GCM and GCM sends the data to the device. If a separate connection would be required for each application in the Android device, then it would drain the battery as well as be expensive in situation where mobile data is limited. So GCM acts as a bridge between application server and the application itself on the device. Users have a choice between whether to get a notification or block a notification for a certain application. Developers have to keep track of the user preferences. If a device is not reachable because it is not connected to the Internet, then the push will be saved at the GCM server for delivery once the device is online. Such push notifications stored at GCM will expire after a certain time.

2.3.2 Apple Push Notification Service

APNS is a robust and highly efficient service for sending information to iOS and OS X devices. This persistent connection enables devices to establish encrypted IP connection that is utilized to receive the notification. When a notification for an application is received while an application is not running, then the iOS operating system installed on the device will alert the user telling about the new data that is ready. The notification message consists of two major pieces of data: the device token and the payload. The device token uniquely identifies a device which is willing to receive the notification while the payload is the actual message that is intended for this device. [10] Figure 6 shows the APNS architecture.

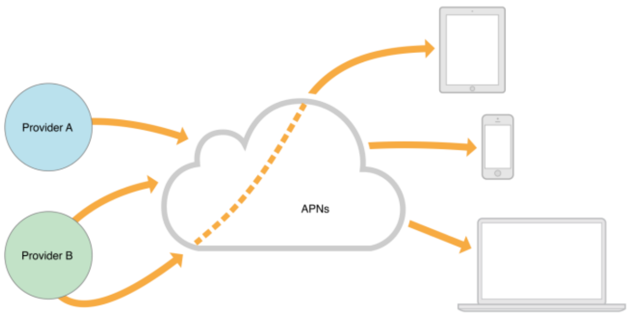
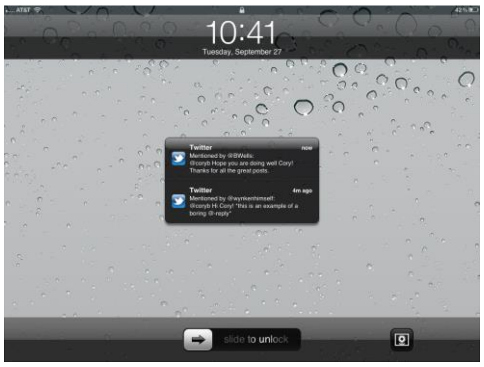


Figure 6. APNs architecture. Reprinted from Apple’s Local and Remote Programming Guide.[10]



Twitter now
Mentioned by @BWells:
@coryb Hope you are doing well Cory! Thanks for all the great posts.

Twitter 4m ago
Mentioned by @wynkenhimself:
@coryb Hi Cory! "this is an example of a boring @-reply"

Figure 7. Push notification received on an iPad for Twitter application

As shown in figure 7 providers are the third-party application servers that establish an encrypted connection with APNs and APNs take the responsibility of sending the notification to the device and the application that the notification is intended for. A notification that is arrived at APNs will be stored if a device is offline and this notification is stored for a limited period of time and delivered once the device is online. This ensures the quality of service (QoS). If multiple notifications are sent to a device then only the latest notification will be sent. The maximum size of a payload is 2vkb and any notification exceeding this limit is discarded. [10] The following JSON object shows a payload in APNs.

```
{
   "aps":{
      "alert":{
         "title":"Game Request",
         "body":"Bob wants to play poker",
         "action-loc-key":"PLAY"
      },
      "badge":5
   }
}
```

List 1.The JSON Payload in APNs

As shown in list 1, this object includes information about the notification title, body and the mode of rendering the notification. Actions that have to be performed after the user opens the application can be also set in the same JSON data.

2.3.3 Windows Push Notification Service

The Windows Push Notification Service (WNS) facilitates developers to send a toast, a small UI feature that pops up on the phone's screen, tile, badge and raw updates from their cloud service to the Windows phone. This is an efficient mechanism to deliver new updates to users without draining the power of the device. Creating push notification features for a Windows phone includes many steps. Figure 8 shows how WNS is implemented.

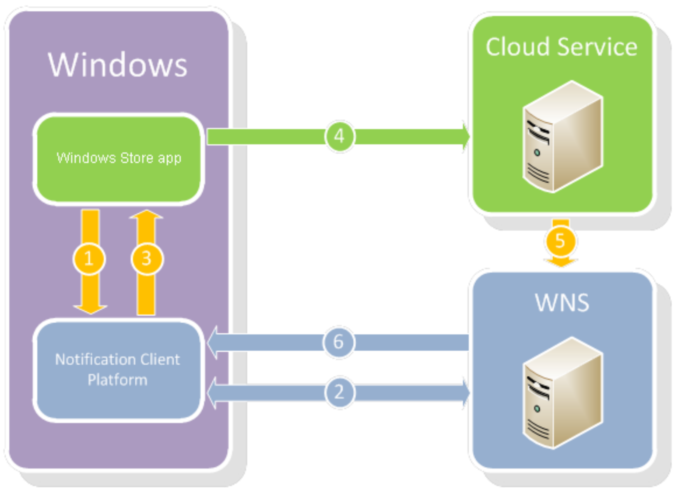


Figure 8. Steps involved in implementing a push notification on Windows phone applications. Reprinted from Windows Development Centre.[11]

Figure 9. Twitter application push notification received on a Windows phone

The first step in sending a push notification includes sending a request for the push notification channel to the Notification Client Platform (NCP). NCP requests WNS to create a notification channel. A Uniform Resource Identifier (URI) is sent back to the requesting device for sending a notification. Windows returns this notification channel to the application. Now the application sends this URI to its cloud service. This URI acts as the call-back interface for the application and its respective cloud service. A developer has the responsibility to implement the call-back mechanism securely. When new data is ready at the server, the application server notifies the WNS using the channel it was provided beforehand in the form of URI. A HTTP POST is issued to WNS that contains a notification payload which is authenticated. WNS routes this notification to appropriate device. A notification channel URI for an application must be requested every time an application starts to make sure that an earlier channel created has not expired. A notification is stored in WNS until the expiration time which can be set by the application cloud service. The Windows phone notification system is not the easiest to handle from a developer point of view because of the complex steps involved. Even though all the notifications are handled through WNS, a separate URI for an individual application is not very friendly compared to Android. [11]

2.4 Cross Platform Push Notification Systems

Third-party push notification service providers are popular among mobile developers because of easiness to manage push notifications via their consoles. Some of the reputed providers include Urban Airship, Parse, and Pushwoosh, Amazon Simple Notification Service etc. Developers do not need to spend time and energy to maintain the notification system for the application which is typically a backend task when using such third-party services. Sending a push notification to a device includes keeping track of device identification and storing each device identification and preferences in a database so as to send a target push to it. A developer has to keep track if a device has permitted to get a notification or not and if a device is online or offline etc. Similarly if an application is uninstalled from a device, the developer has to clear the data associated with the device in order to reduce the overload on the back from the database. The push notification works much like a subscribe publish model where a client subscribes for information and the information is published to those clients who have subscribed. So it is important to keep track of the user's interests to receive notifications. If an application sends out an unnecessary push notification to the device that explicitly unsubscribed to push notifications via their application manager console in the device, it is very likely that the user will uninstall the application or use an alternative application. User experience is a determining factor in the push notification system because all the applications installed in a user device are competing to make the user spend as much time as possible in them. [12] These factors contribute to the use of a third-party push notification system.

Figure 10 below shows the common procedure involved when implementing a third-party push notification system such as Urban Airship.

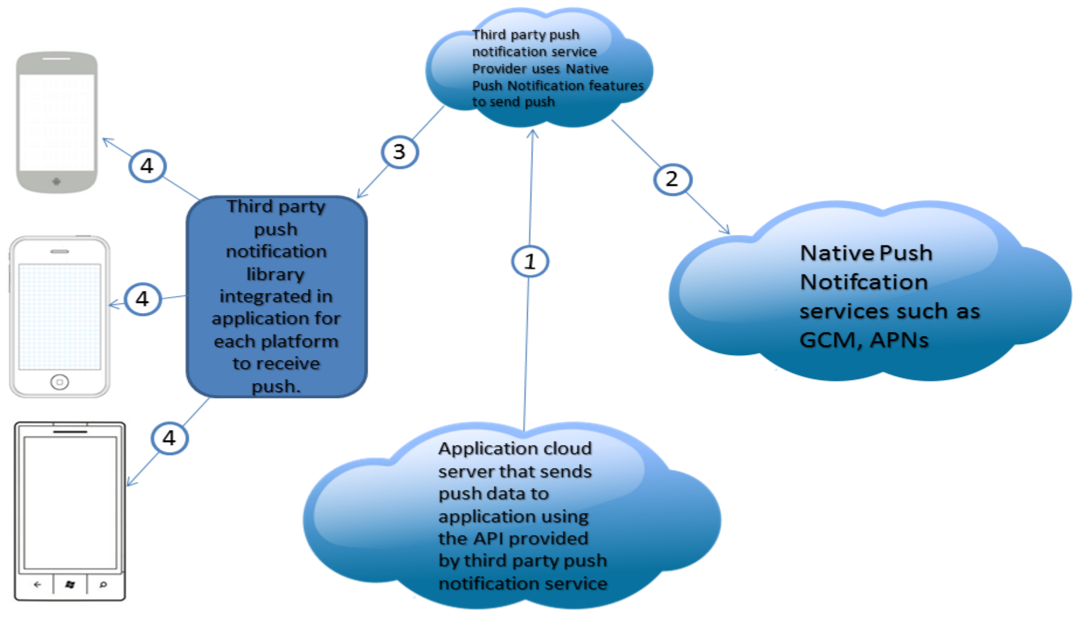
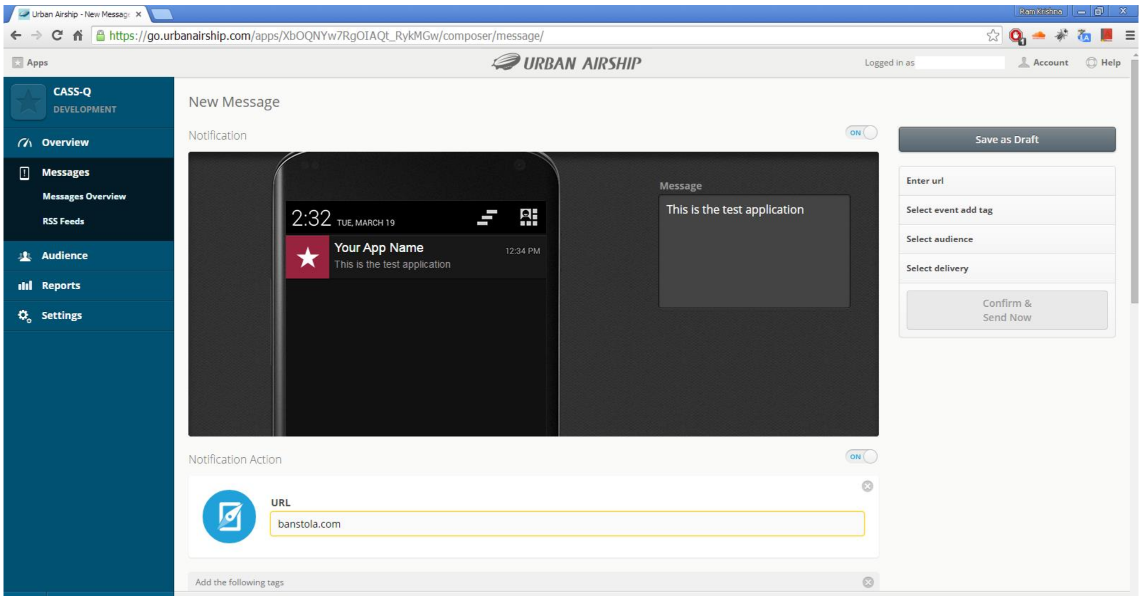

Third party push notification service Provider uses Native Push Notification features to send push
3
2
4
Third party push notification library integrated in application for each platform to receive push.
4
1
Native Push Notifcation services such as GCM, APNs
4
Application cloud server that sends push data to application using the API provided by third party push notification service

Figure 10. Architecture of commonly used third-party push notification systems.

As shown in figure 10, the client application uses the library provided by the notification service providers to update the preferences of the user to receive notifications and handle the payload once the push is received. Developers can utilize this payload to create custom notifications using a toast, badge or status bar notification based on the platform. When a new push is ready at the server of the application a call will be made to the notification service to handle the payload and the group of devices that have agreed to receive this certain push across different platforms. The notification service works as middle man to handle this push using each platform's native push notification service and sends the payload to each device and then finally to the target application. Such an interface between the native notification systems makes it easier for developers to manage the push notification.



https://go.urbanairship.com/apps/XbOQNYw7RgOIAQt_RykMGw/composer/message/

Message: This is the test application

URL: banstola.com

Figure 11. Push notification console of Urban Airship.

Figure 11 shows the push notification console of Urban Airship Push Notification System which can be used to directly send push notifications to the application. The developer can customize to send rich text messages. Similarly many behaviours of the push notification can be modified from this console directly e.g. opening a browser to take the user to a certain website once the user taps the notification. It makes easier for enterprises to compose notifications easily to reach their audiences.

2.5 WebSocket Notification

All the discussed push notification technologies are capable of sending push to native applications for the platform in question. As HTML5, JavaScript and CSS have evolved, web applications commonly known as webapps are gaining popularity among web developers. jQuery Mobile, AngularJS etc. are popular JavaScript frameworks for creating webapps that have rich user interfaces. Since webapps run on the web browser they lack the capability to receive native push notifications from server because there is no way the server can contact a client unless initiated by a client. A perpetual connection between the webapps and their respective servers must be maintained for getting any new information available on the server which is only limited within the scope of the webapps. There are technologies such as polling and long polling where the server is continuously contacted for any new information over a period of time or establishing a WebSocket which has a full bi-directional communication capability over a TCP

(Transmission Control Protocol). The later, WebSocket communication, behaves like a true push notification system for web applications.

A new specification is under way to display notifications outside the context of a web page. The development of Application Programming Interface (API) for notifications is under way by W3C, World Wide Web Consortium [13]. Such API would enable developers to utilize the native-like push notification system that can alert the users outside the context of the web pages. Polling is a pseudo push notification technique because the core concept of push notification is a server being able to push data to the client but not client asking for any new data in the server.

WebSocket is the future of push notifications for web applications. IETF (Internet Engineering Task Force) created the first specification of web the socket in 2011. According to the specification, the WebSocket protocol enables bidirectional communications between a client that is running a trusted code in a controlled environment to remote host model use for this is the origin-based security model commonly used by web browsers [14]. HTTP long polling has a high overhead because the server is forced to use different underlying TCP connection for each client.

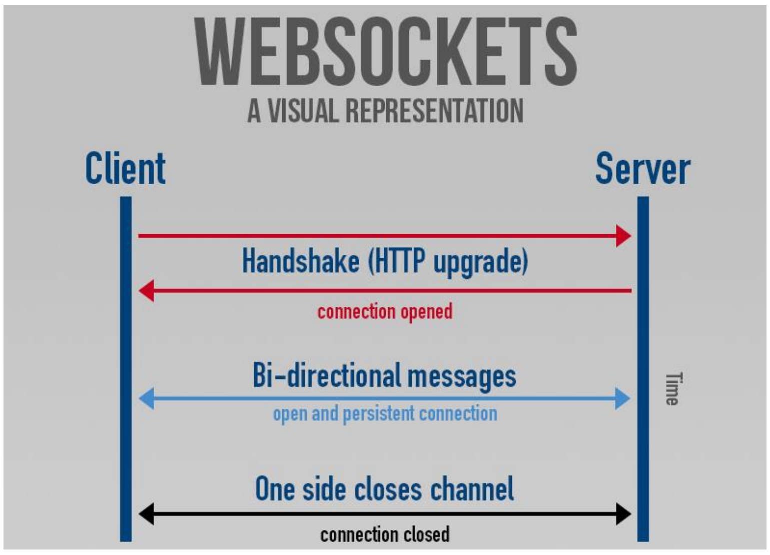


Figure 12. Bidirectional communication with web socket. Reprinted from PubNub.com [15].

As shown in figure 12, the client initiates a handshake request to the server and the server returns the WebSocket handshake in response. After this handshake a persistent and open bi-directional connection is established for data exchange. This connection can be closed by either side. Below is an example of a handshake from a client and server.

Client handshake	Server handshake
GET /chat HTTP/1.1 Host: server.example.com Upgrade: WebSocket Connection: Upgrade Sec-WebSocket-Key:dGhlIHNhbXBsZSBub25jZQ== Origin: http://example.com Sec-WebSocket-Protocol: chat, superchat Sec-WebSocket-Version: 13	HTTP/1.1 101 Switching Protocols Upgrade: websocket Connection: Upgrade Sec-WebSocket:s3pPLMBiTxaQ9kYGzzhZRbK+xOo= Sec-WebSocket-Protocol: chat

Table 1. Client-Server handshake during a WebSocket connection

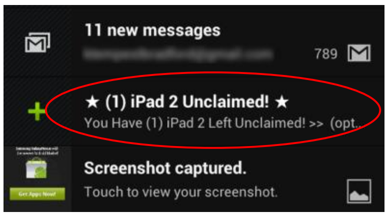
The primary difference between WebSocket communication and traditional communication is that once the client and server agree to exchange data between them after the handshake; they can asynchronously send data to each other. This is not the case in the case of HTTP connection where client makes a request for every new piece of data it needs making the traffic size bigger with Meta data that is needed for client-server communication [15].

2.6 Security Risks of Push Notification Services

Internet communication security is a sensitive topic at present among individuals and organisations because the aftermath of a data theft is very expensive, both financially and socially. There are growing concerns about how push notification systems implemented using WebSocket, GCM etc. could be exploited by hackers to send malicious code and steal user data. Since little research has been done in the vulnerability of push services, even reputable services like GCM and Amazon Device Messaging (ADM) can be exploited to steal sensitive message from the target device, wrongfully install or uninstall an application, lock the legitimate user to prevent the user from using the device as well as wipe the whole device. [16]

In GCM, the developers can willingly program the application that steals user information, monitor user behaviour, install unwanted applications in the user's device etc. On a report published by Computer World, a leading technology magazine, citing a research done at

Kaspersky Lab states cybercriminals are controlling the malware applications, short for malicious programs, installed on an Android phone using GCM for stealing user data. The malicious application in question is Trojan-SMS.AndroidOS.FakeInst.a which is capable of sending text messages to premium rate numbers, delete incoming messages without the knowledge of the phone owner, generate shortcuts to malicious sites and display the notifications advertising other malicious programs as useful app or games. Such Trojan applications affected millions of Android mobile devices. [17] Despite security threats spammers utilize the push notification service to send advertisements, fake links etc. Figure 13 shows an example of a spamming notification alert from an application on an Android phone.



11 new messages
[redacted] 789

★ (1) iPad 2 Unclaimed! ★
You Have (1) iPad 2 Left Unclaimed! >> (opt...

Screenshot captured.
Touch to view your screenshot.

Figure 13. Spam push notification message on a Android Phone

WebSocket is an emerging technology which has been utilized rapidly for receiving push notifications in web applications. Security vulnerability involved in WebSocket has not been studied in detail. Web services security depends on Transport Layer Security (TLS) encryption and the same-origin policy. WebSocket differs from the HTTP protocol in terms of origin policy. Server makes the decision to allow or deny a connection during the handshake; this can be easily spoofed. A malicious website is able to set up remote shell access which can eventually take control of the victim's web browser in real time. Security should be taken into consideration while implementing a push notification system in order to prevent any backdoors for hackers. [18]

3 Analysis, Development and Deployment

CASS has two clients for the research subject to submit the answered queries namely Browser client and Native client (Android application). Of the four research method discussed in the introduction section of this paper, the randomized data collection method uses either of the notification since the time to answer a query is randomly generated by the CASS system, it is important to notify a subject once the query is ready. The researcher involved in creating the research aka administrator can choose between which types of notification to send to a particular subject. Entering a phone number implies that that the research subject is able to receive a mobile push notification when a query is ready to answer while entering an email address implies the notification will be sent using an email when a query is ready. This gives administrator the flexibility to choose a notification system based on the respondent's choice of client.

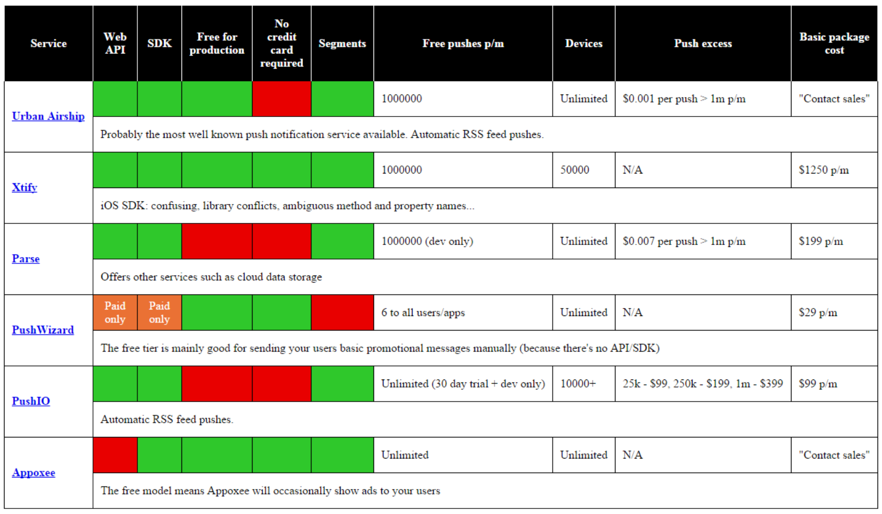
3.1 Necessity and Comparisons between Nnotification Systems

Push notification service providers, both native and third-party, perform the common task of sending push notifications to mobile applications installed on user devices. The push payload adds value to the user and has to be context sensitive, e.g. a football enthusiast who uses an application to monitor the score in a game cares about the most recent score or event for an ongoing match but not about the information for the previous game. In this competitive era of mobile applications, each application installed on user device is competing against others to grab user attention and make the user spend more time on them.

All mobile platforms in existence have their own notification service. But third-party notification service providers are gaining a momentum among developers. So, a primary question arises about why developers choose a third-party notification service providers when each platform has its own notification service? The answer lies in the complexity of managing the push notification services. In addition to the easy management of push notifications, these providers' offer analytics that help businesses understand the effectiveness of the push. Developers can track the number of the push sent, applications opened response to application as well as the amount of time spent on the application. Such information is vital for understanding the need of the application user. At present, we live in a technological era where a ready-made solutions that work on the *plug and play* manner are favoured over creating something from scratch unless absolutely necessary.

The number of push that have to be sent every day differs from application to application depend upon the number of active users. So, it is not a wise decision to implement native push notification for all applications. For example the BBC news (British Broadcasting Corporation) mobile application sends several push notifications per day to a user depending upon user preference. If a user has subscribed to getting a notification for breaking news or some other news of his/her interest, then he/she might get frequent notifications and developers have to write algorithms to manage each devices individually. But since BBC news has such a large numbers of users , it is a better for them to implement a native push notification service because it is more cost-friendly and they have resources to manage notifications, while applications like CASS sends notifications on a certain time of the day and for a certain period of time. CASS has a very limited number of users and primary use of the application is for collecting context-sensitive user data. From CASSs use case point of view, a third-party notification service provider is better for number a number of reasons. The primary reason is the multiplatform approach of the third-party providers. The same set of instructions on a server can be used to send push notifications to iOS and Android devices. So it eliminates the need to write different backend code for different platforms. CASS notifications are for a certain period of time for a user. So, it is not effective spending time and resources to make custom notification servers for push. A push notification is a vital part of CASS as data collection is time-sensitive and outsourcing push notification gives developers time to focus on other important aspects of CASS.

Third-party providers come with tools to assist developers for deployment, testing and debugging. The reasons why UrbanAirship was selected for the CASS push notification while there are a number of other providers is discussed in this chapter in detail by making comparisons between the other third-party providers. Figure 14 provides an overview of popular providers along with their pros and cons.



Service	Web API	SDK	Free for production	No credit card required	Segments	Free pushes p/m	Devices	Push excess	Basic package cost
Urban Airship	🟩	🟩	🟩	🟥	🟩	1000000	Unlimited	$0.001 per push > 1m p/m	"Contact sales"
	Probably the most well known push notification service available. Automatic RSS feed pushes.								
Xtify	🟩	🟩	🟩	🟩	🟩	1000000	50000	N/A	$1250 p/m
	iOS SDK: confusing, library conflicts, ambiguous method and property names...								
Parse	🟩	🟩	🟥	🟥	🟩	1000000 (dev only)	Unlimited	$0.007 per push > 1m p/m	$199 p/m
	Offers other services such as cloud data storage								
PushWizard	Paid only	Paid only	🟩	🟩	🟥	6 to all users/apps	Unlimited	N/A	$29 p/m
	The free tier is mainly good for sending your users basic promotional messages manually (because there's no API/SDK)								
PushIO	🟩	🟩	🟥	🟥	🟩	Unlimited (30 day trial + dev only)	10000+	25k - $99, 250k - $199, 1m - $399	$99 p/m
	Automatic RSS feed pushes.								
Appoxee	🟥	🟩	🟩	🟩	🟩	Unlimited	Unlimited	N/A	"Contact sales"
	The free model means Appoxee will occasionally show ads to your users								

Figure 14. Comparison between third-party push notification providers. Reprinted from b2bCloud.com.[19]

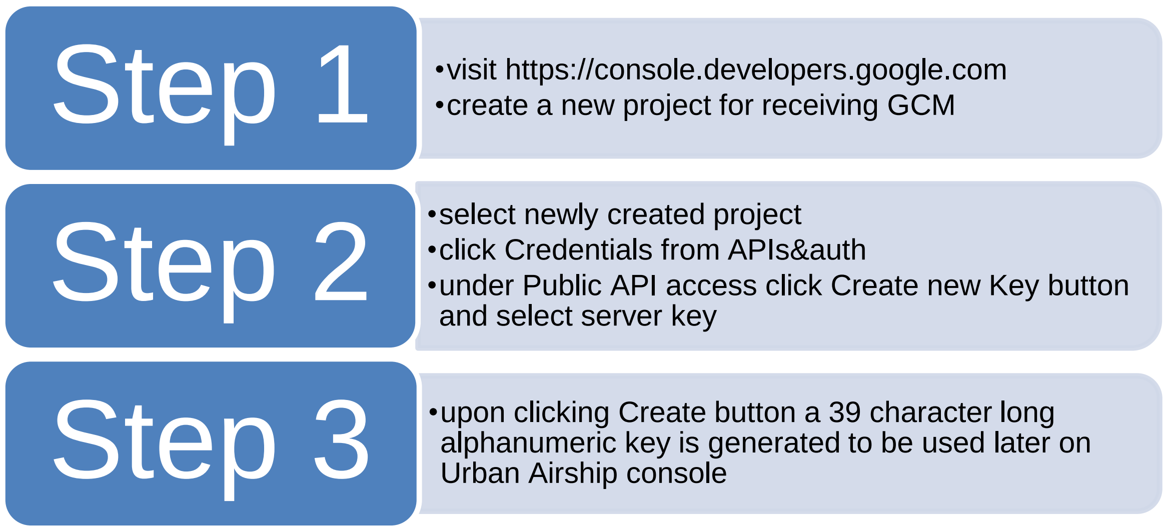
Based on figure 19, popular providers include Urban Airship, Xitfy, Parse, PushIO, Pusher etc. Amazon SNS has emerged as a new player in the push notification business since it is now possible to send mobile push notifications with it. Parse is one the most successful provider after its purchase by Facebook. Many recognized brands like

Samsung, EBay, and Volvo have outsourced their push service to Parse. Urban Airship is cheaper than other providers as well as being a trusted brand and reliable. There is no restriction on the number of devices and the number of push that can be sent unlike some other vendors. Also cost is based on the number of push sent. Cost per push is cheaper than other vendors. Segmentation of the user group is easier with their web API. Developers can target users based on a geo location. Managing the push subscribers is easier with Urban Airship's platform specific libraries. On the other hand other competitors are either unreliable or in their nascent stage of development in this area [19]. WebSocket push with Pusher is an alternative solution to using Urban Airship but it would require managing the subscribed devices, which requires more backend programming. Urban Airship being a reliable player in the mobile push notification system with the possibility of easy integration with applications developed for popular mobile platforms, was selected for the CASS notification purpose.

3.2 Implementation of Urban Airship Push Notification System

3.2.1 GCM and Urban Airship Setup

Urban Airship utilizes platform-specific notification service to send push notifications. So the credentials from each platform in which application will be deployed should be provided to the Urban Airship console. This thesis primarily focuses on push implemented for the Android version of CASS. The procedure to implement credentials from Google is described below in three steps.



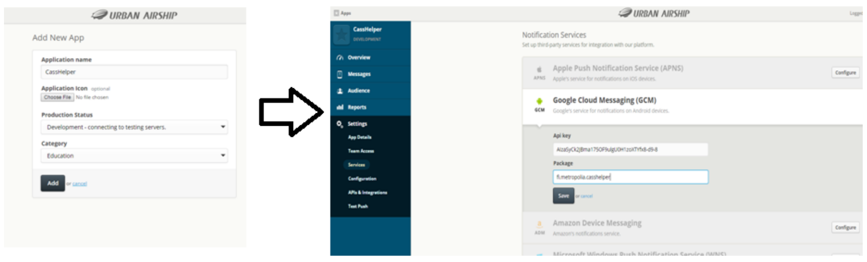
Step 1
- visit https://console.developers.google.com
- create a new project for receiving GCM

Step 2
- select newly created project
- click Credentials from APIs&auth
- under Public API access click Create new Key button and select server key

Step 3
- upon clicking Create button a 39 character long alphanumeric key is generated to be used later on Urban Airship console

Figure 15. User specification for CASS push notification

When a project is created at Google console, it will assigned a unique project number which is required in the client application for the push notification channel.



Application name: CassHelper
Application Icon optional
Production Status: Development - connecting to testing servers.
Category: Education

Notification Services
Set up third-party services for integration with our platform.
Apple Push Notification Service (APNS) — Apple's service for notifications on iOS devices.
Google Cloud Messaging (GCM) — Google's service for notifications on Android devices.
Api key: AIzaSyCk2jBma17SOF9ulgU0H1zoXTYfx8-d9-8
Package: fi.metropolia.casshelper
Amazon Device Messaging — Amazon's notifications service.
Microsoft Windows Push Notification Service (WNS)

Figure 16. Creating a new project at Urban Airship and providing API key for GCM

Since GCM is the transport method that Urban Airship uses to deliver the push notification, the GCM API key should be provided to the Urban Airship console. As shown in the figure 15 each platform has their own mechanism to provide the API key for push notification and those provided key must be given to Urban Airship. Three keys are generated for an application at the Urban Airship console and their use is as shown in table. [20]

Table 2. Urban Airship generated string keys.[20]

Urban Airship key	Use
App key	Identifying the application setup. Used in application that receives the push.
App Secret	Identifying the application setup secret. Used in the client application.
App Master Secret	Server to server API access key. Used in the client server that sends the push payload.

As shown in table 2, there are three different keys generated by Urban Airship when a new application is registered for API access, in order to send a successful push.

3.2.2 Urban Airship Backend Implementation

PHP is the server-side language used in CASS. When new research is created in CASS new entries are made to a file that runs a certain PHP file at the exact time when a query is ready. Cron is a daemon that runs persistently in the background of a Linux computer, in this case Linux server, and it performs the job of automatically running the tasks according to a schedule. The schedule is contained in a file called crontab. Cron command does this job of executing a specific command which in turn executes a certain file [21]. The timestamp and action is provided to crontab when a new research is created,i.e. new entries are made to the pre-existing schedule table. For example, when a researcher creates a new research with a unique research identification number, then a new crontab entry is added which has the task to execute notifications.php?id=research_id file on the server. The data associated with the research are fetched using the research identification number.

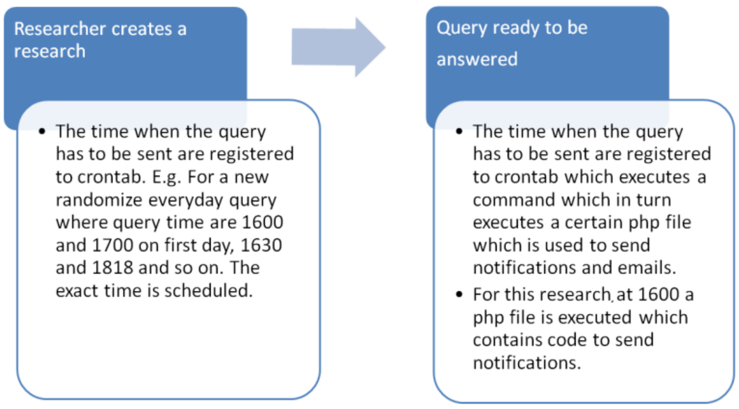

Researcher creates a research
The time when the query has to be sent are registered to crontab. E.g. For a new randomize everyday query where query time are 1600 and 1700 on first day, 1630 and 1818 and so on. The exact time is scheduled.
Query ready to be answered
The time when the query has to be sent are registered to crontab which executes a command which in turn executes a certain php file which is used to send notifications and emails.
For this research, at 1600 a php file is executed which contains code to send notifications.

Figure 17. Scheduling query and notification in CASS using cron

As the PHP file is executed at the predefined time when a query has to be delivered to a research subject, as shown in figure 16, server side implementation of both the email notification and Urban Airship notification is done at this file. A PHP class is created in a separate PHP file that contains a function to send JSON data to Urban Airship for notification when a new query is ready. Each device is recognized using a token for specification. The 12 character long unique alphanumeric token is randomly generated once a new research subject is created in CASS research admin console and is stored in the CASS database.

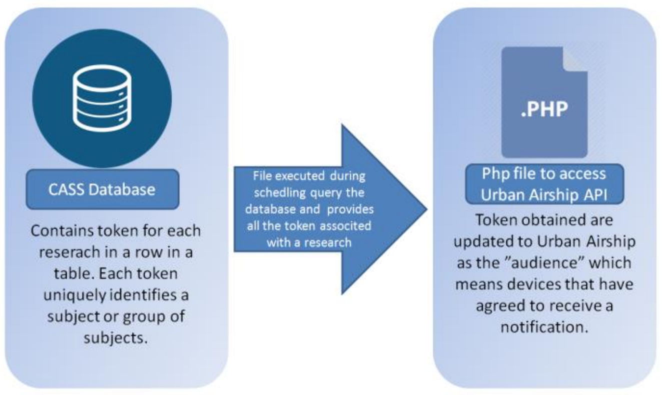
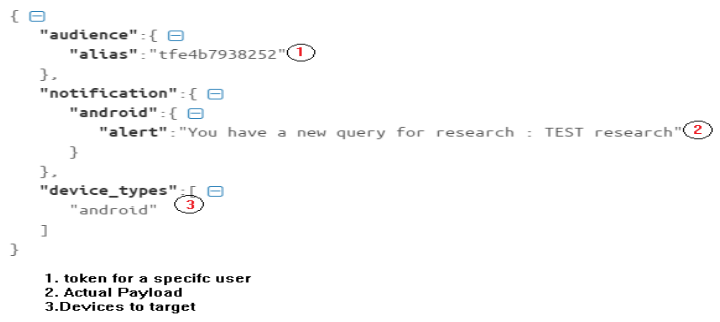

CASS Database
Contains token for each reserach in a row in a table. Each token uniquely identifies a subject or group of subjects.
File executed during schedling query the database and provides all the token associted with a research
.PHP
Php file to access Urban Airship API
Token obtained are updated to Urban Airship as the "audience" which means devices that have agreed to receive a notification.

Figure 18. User specification for CASS push notification

Figure 18 demonstrates how user specification is done at CASS so that push notifications will be sent to the right user at the right time.



```
{
   "audience":{
      "alias":"tfe4b7938252" 1
   },
   "notification":{
      "android":{
         "alert":"You have a new query for research : TEST research" 2
      }
   },
   "device_types":[
      "android" 3
   ]
}
```

1. token for a specifc user
2. Actual Payload
3.Devices to target

Figure 19. Example of JSON data sent to Urban Airship

After the user specification is done, data along with payload will be coded in JSON format which includes information such as the target device and research name. This JSON data is sent via API for push notification. The task of handling the push notification to specific device and user based on the JSON data is done by Urban Airship. This way of handling push using Urban Airship is much easier. The task of checking which user has allowed the push, handling of push if the client is offline and managing the token is done at Urban Airship which is not the case if GCM is used. GCM is a transport method while Urban Airship is the platform to handle push notification.

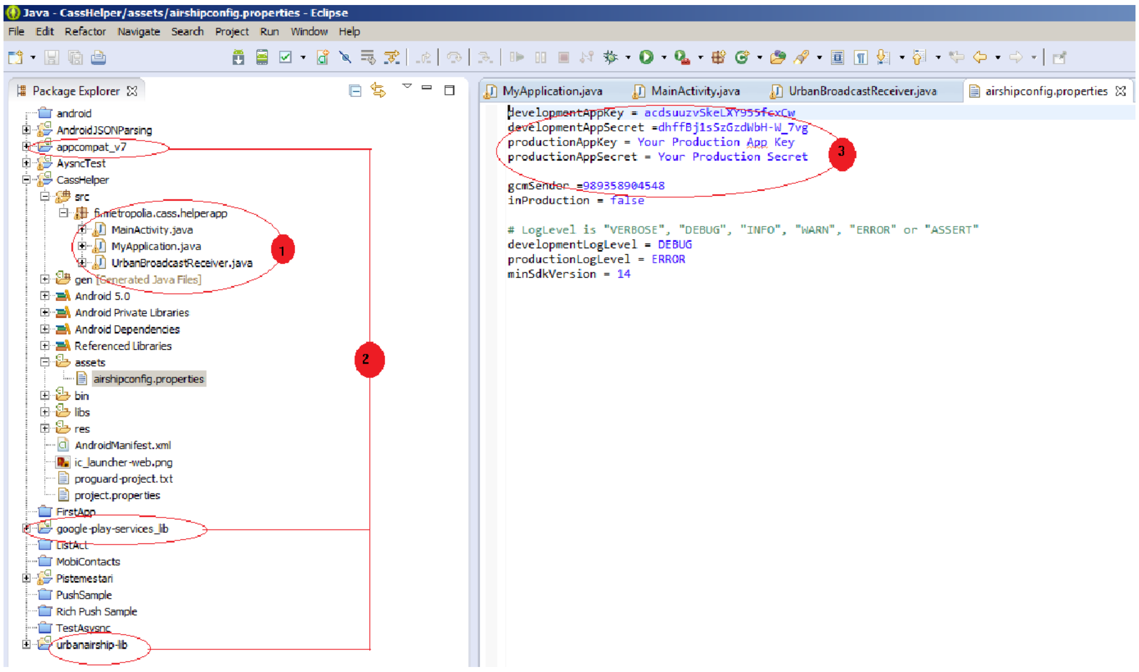
3.2.3 Client-Side Implementation

A separate application was developed for receiving notifications when a query is ready to be answered by using the CASS-Q Android application for testing purposes namely CassHelper. CassHelper is a lightweight Android application that takes a 12 character long unique alphanumeric token as its input for managing push and should be installed on the device along with the CASS-Q application. As both applications need to have a common token. The CASS-Q application requires a token to uniquely identify each research subject while CassHelper requires token to specify the devices that have subscribed to notification. Once a token is entered in the CassHelper application, it will be immediately updated at the Urban Airship server as audience.

There are three different Java classes for performing different tasks in the CassHelper application namely MainActivity, MyApplication and UrbanBroadcastReceiver. The task of enabling the push, setting up a channel to connect to Urban Airship and configuring for first time use is done at MyApplication class. This channel the remains same until the application is uninstalled from the user device. Various information such as the time spent in the application, if the push is clicked or not and other analytics data are sent to Urban Airship for analysing the effectiveness of the notification through this channel. Visual information of such data can be viewed at the Urban Airship console. MainActivity class handles updating the token and rendering the user interface. It is the entry point when the application runs every time.

A broadcast receiver is an Android component that listens to particular events and performs actions when the events are triggered. Otherwise it is dormant [22]. A broadcast receiver implemented at CassHelper receives the notification data which contains information about the research and notification is set up and upon opening the status bar notification CASS-Q application is launched where a subject can answer a query that is ready. UrbanBroadcastReceiver class extends the BroadcastReciever class of Android which overrides the onReceive method of BroadcastReciever. When a push is received, then actions that have to be presented to the user are implemented at onRecieve method.

Google Play library, Urban Airship library and Support library version seven have to be included in the Android Project of CassHelper. There is a configuration file in the asset folder of the project where API key namely AppKey and AppSecret from Urban Airship as well as project number obtained from Google Console should be added to configuration file as shown in figure 20.


Java - CassHelper/assets/airshipconfig.properties - Eclipse
File Edit Refactor Navigate Search Project Run Window Help
Package Explorer
MyApplication.java
MainActivity.java
UrbanBroadcastReceiver.java
airshipconfig.properties
1
2
3

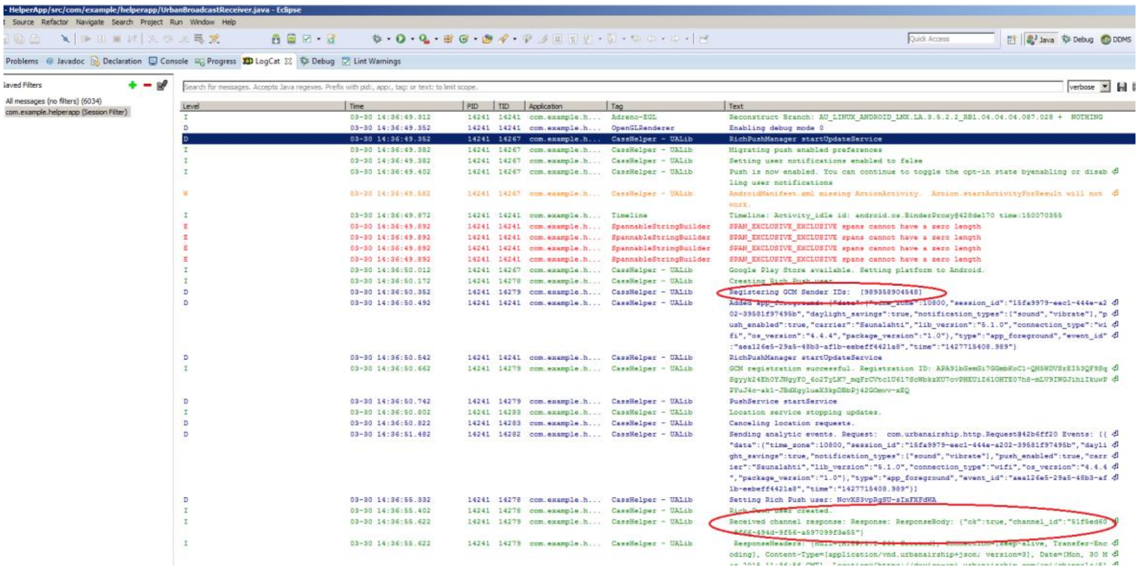
```
android
AndroidJSONParsing
appcompat_v7
AysncTest
CassHelper
  src
    fi.metropolia.cass.helperapp
      MainActivity.java
      MyApplication.java
      UrbanBroadcastReceiver.java
  gen [Generated Java Files]
  Android 5.0
  Android Private Libraries
  Android Dependencies
  Referenced Libraries
  assets
    airshipconfig.properties
  bin
  libs
  res
  AndroidManifest.xml
  ic_launcher-web.png
  proguard-project.txt
  project.properties
FirstApp
google-play-services_lib
ListAct
MobiContacts
Pistemestari
PushSample
Rich Push Sample
TestAsysnc
urbanairship-lib
```

```
developmentAppKey = acdsuuzvSkeLXY955fcxCw
developmentAppSecret =dhffBj1sSzGzdWbH-W_7vg
productionAppKey = Your Production App Key
productionAppSecret = Your Production Secret

gcmSender =989358904548
inProduction = false

# LogLevel is "VERBOSE", "DEBUG", "INFO", "WARN", "ERROR" or "ASSERT"
developmentLogLevel = DEBUG
productionLogLevel = ERROR
minSdkVersion = 14
```

Figure 20. CassHelper development setup in Eclipse Application Development Tool(ADT)


HelperApp/src/com/example/helperapp/UrbanBroadcastReceiver.java - Eclipse
Source Refactor Navigate Search Project Run Window Help
Quick Access
Java
Debug
DDMS
Problems
Javadoc
Declaration
Console
Progress
LogCat
Debug
Lint Warnings
Saved Filters
All messages (no filters) (6034)
com.example.helperapp (Session Filter)
Search for messages. Accepts Java regexes. Prefix with pid:, app:, tag: or text: to limit scope.
verbose

```
Level  Time                PID    TID    Application          Tag                     Text
I      03-30 14:36:49.312  14241  14241  com.example.h...     Adreno-EGL              Reconstruct Branch: AU_LINUX_ANDROID_LNX.LA.3.5.2.2_RB1.04.04.04.087.028 +  NOTHING
D      03-30 14:36:49.352  14241  14241  com.example.h...     OpenGLRenderer          Enabling debug mode 0
D      03-30 14:36:49.352  14241  14267  com.example.h...     CassHelper - UALib      RichPushManager startUpdateService
I      03-30 14:36:49.382  14241  14267  com.example.h...     CassHelper - UALib      Migrating push enabled preferences
I      03-30 14:36:49.382  14241  14267  com.example.h...     CassHelper - UALib      Setting user notifications enabled to false
I      03-30 14:36:49.402  14241  14267  com.example.h...     CassHelper - UALib      Push is now enabled. You can continue to toggle the opt-in state byenabling or disab
                                                                                      ling user notifications
W      03-30 14:36:49.602  14241  14267  com.example.h...     CassHelper - UALib      AndroidManifest.xml missing ActionActivity.  Action.startActivityForResult will not
                                                                                      work.
I      03-30 14:36:49.872  14241  14241  com.example.h...     Timeline                Timeline: Activity_idle id: android.os.BinderProxy@428de170 time:150070355
E      03-30 14:36:49.892  14241  14241  com.example.h...     SpannableStringBuilder  SPAN_EXCLUSIVE_EXCLUSIVE spans cannot have a zero length
E      03-30 14:36:49.892  14241  14241  com.example.h...     SpannableStringBuilder  SPAN_EXCLUSIVE_EXCLUSIVE spans cannot have a zero length
E      03-30 14:36:49.892  14241  14241  com.example.h...     SpannableStringBuilder  SPAN_EXCLUSIVE_EXCLUSIVE spans cannot have a zero length
E      03-30 14:36:49.892  14241  14241  com.example.h...     SpannableStringBuilder  SPAN_EXCLUSIVE_EXCLUSIVE spans cannot have a zero length
I      03-30 14:36:50.012  14241  14267  com.example.h...     CassHelper - UALib      Google Play Store available. Setting platform to Android.
I      03-30 14:36:50.172  14241  14278  com.example.h...     CassHelper - UALib      Creating Rich Push user.
D      03-30 14:36:50.352  14241  14279  com.example.h...     CassHelper - UALib      Registering GCM Sender IDs:  [989358904548]
D      03-30 14:36:50.492  14241  14241  com.example.h...     CassHelper - UALib      Added app_foreground: {"data":{"time_zone":10800,"session_id":"15fa9979-eec1-444e-a2
                                                                                      02-39581f97495b","daylight_savings":true,"notification_types":["sound","vibrate"],"p
                                                                                      ush_enabled":true,"carrier":"Saunalahti","lib_version":"5.1.0","connection_type":"wi
                                                                                      fi","os_version":"4.4.4","package_version":"1.0"},"type":"app_foreground","event_id"
                                                                                      :"aea126e5-29a5-48b3-af1b-eebeff4421a8","time":"1427718408.989"}
D      03-30 14:36:50.542  14241  14241  com.example.h...     CassHelper - UALib      RichPushManager startUpdateService
I      03-30 14:36:50.662  14241  14279  com.example.h...     CassHelper - UALib      GCM registration successful. Registration ID: APA91bGemGi7GGmbKoC1-QN5WDVSrEI53QF9Sq
                                                                                      Sgyyk24Eh0YJNgyF0_6o2TyLK7_mqFrCVtc1U617ScWbkzXU7ovPHEU1Z610HTE07h8-mLV9IWGJihiIkuwP
                                                                                      PYuJ4c-ak1-JBdXgylueX3kpDBbPj42GOmvv-xEQ
D      03-30 14:36:50.742  14241  14279  com.example.h...     CassHelper - UALib      PushService startService
I      03-30 14:36:50.802  14241  14283  com.example.h...     CassHelper - UALib      Location service stopping updates.
D      03-30 14:36:50.822  14241  14283  com.example.h...     CassHelper - UALib      Canceling location requests.
D      03-30 14:36:51.482  14241  14282  com.example.h...     CassHelper - UALib      Sending analytic events. Request:  com.urbanairship.http.Request@42b6ff20 Events: [{
                                                                                      "data":{"time_zone":10800,"session_id":"15fa9979-eec1-444e-a202-39581f97495b","dayli
                                                                                      ght_savings":true,"notification_types":["sound","vibrate"],"push_enabled":true,"carr
                                                                                      ier":"Saunalahti","lib_version":"5.1.0","connection_type":"wifi","os_version":"4.4.4
                                                                                      ","package_version":"1.0"},"type":"app_foreground","event_id":"aea126e5-29a5-48b3-af
                                                                                      1b-eebeff4421a8","time":"1427718408.989"}]
D      03-30 14:36:55.332  14241  14278  com.example.h...     CassHelper - UALib      Setting Rich Push user: NcvXS3vpRgSU-sIxFXFdWA
I      03-30 14:36:55.402  14241  14278  com.example.h...     CassHelper - UALib      Rich Push user created.
I      03-30 14:36:55.622  14241  14279  com.example.h...     CassHelper - UALib      Received channel response: Response: ResponseBody: {"ok":true,"channel_id":"51f5ed60
                                                                                      -9f66-494d-9f56-a597099f3e55"}
I      03-30 14:36:55.622  14241  14279  com.example.h...     CassHelper - UALib       ResponseHeaders: {null=[HTTP/1.1 200 Created], Connection=[keep-alive, Transfer-Enc
                                                                                      oding], Content-Type=[application/vnd.urbanairship+json; version=3], Date=[Mon, 30 M
```

Figure 21. Log of CassHelper running for first time after installation.

As shown in figure 21, GCM registration, channel creation etc. is done when the application is running for the first time.

Notification, email notification for a browser client or push notification for a native client , is a must for CASS because of the random time in which query are delivered to subjects that researchers needs to gather from a subject in a randomized data collection method. Unlike other data collection methods in the randomized method the delivery time of query differs every day. So notifying a subject is a reliable method to make sure a subject is reminded to answer a query. At present CASS-Q and CassHelper act as separate applications. When a query is ready at CASS server, the notification is sent from the server to Urban Airship as described in the section 3.3.1 and it will further send the notification to CassHelper.

3.3 Email Notification Implementation

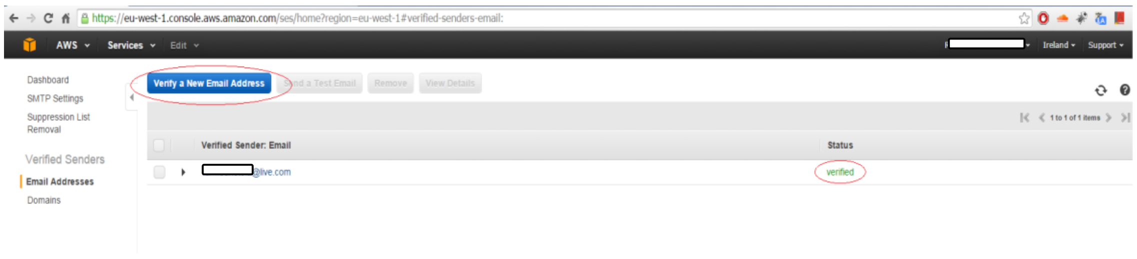
3.3.1 Amazon Simple Email Service

Amazon's Simple Notification Service (SNS) was used in CASS for email notification before custom email for CASS was implemented. It is a paid service and developers do not have options for sending customized email. CASS is hosted in Amazon's Elastic Cloud server which allows developers to utilize another feature known as Amazon's Simple Email Service (SES). For small systems like CASS, the free tier offered by Amazon's is enough. Even when the numbers of emails exceed the monthly limit of 62 thousand emails, the cost will be paid only for the emails that are sent.

SES implementation optimizes the percentage of emails that are delivered successfully, email server management, network configuration and IP address reputation. Today's email clients are smart to filter spam emails but sending emails using SES decreases the chance of ending an email to the junk folder. Similarly Internet Service Providers (ISP) has systems that blocks the IP addresses that sends spam emails regularly. In such process of filtering email a legitimate email might end of not reaching the destination. The decision to deliver an email by ISP is made based on the origination of the email. SES has a quality control system to ensure high quality emails to be delivered from their IP which makes Amazon SES as a trusted email origin. [23]

Before any email can be sent using SES, the sender email address has to be verified in the Amazon Console. An email is sent to the sender’s address for verification and once the verification is done, the email address will be eligible as a sender. A SMTP user has to be created at Amazon Console to send notification emails in CASS using SMTP. A username and a password are automatically generated for outbound emails. Since

CASS does not require any incoming emails, an open source php library for sending emails using SMTP namely PHPMailer is used for this purpose.



https://eu-west-1.console.aws.amazon.com/ses/home?region=eu-west-1#verified-senders-email:

AWS Services Edit [redacted] Ireland Support

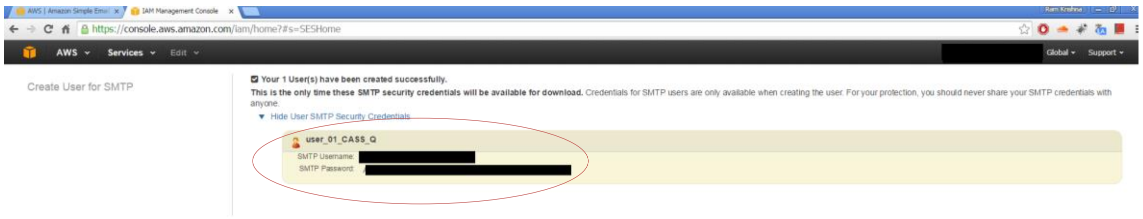
Dashboard
SMTP Settings
Suppression List Removal
Verified Senders
Email Addresses
Domains

Verify a New Email Address | Send a Test Email | Remove | View Details

1 to 1 of 1 items

Verified Sender: Email	Status
[redacted]@live.com	verified

Figure 22. Sender email address verification at Amazon console.



https://console.aws.amazon.com/iam/home?#s=SESHome

AWS Services Edit [redacted] Global Support

Create User for SMTP

Your 1 User(s) have been created successfully.

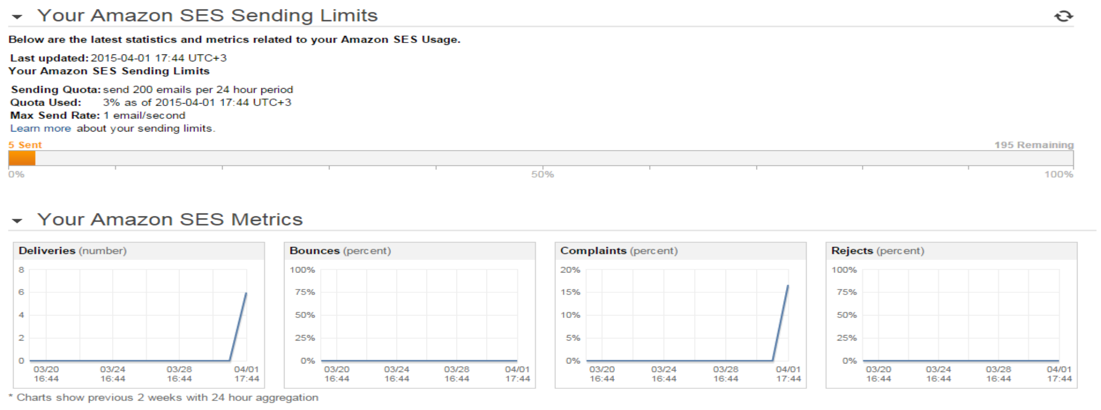
This is the only time these SMTP security credentials will be available for download. Credentials for SMTP users are only available when creating the user. For your protection, you should never share your SMTP credentials with anyone.

Hide User SMTP Security Credentials

user_01_CASS_Q
SMTP Username: [redacted]
SMTP Password: [redacted]

Figure 23. Generating username and password for sending emails using PHPMailer

Figure 23 shows a verified email address at Amazon console. This approach of verifying the email address is useful to maintain the origin of high quality emails. The credentials generated as shown in figure 23 are used to access the SMTP server at Amazon.



Your Amazon SES Sending Limits

Below are the latest statistics and metrics related to your Amazon SES Usage.

Last updated: 2015-04-01 17:44 UTC+3
Your Amazon SES Sending Limits

Sending Quota: send 200 emails per 24 hour period
Quota Used: 3% as of 2015-04-01 17:44 UTC+3
Max Send Rate: 1 email/second
Learn more about your sending limits.

5 Sent — 195 Remaining

Your Amazon SES Metrics

* Charts show previous 2 weeks with 24 hour aggregation

Figure 24. Line graph to visualize email success metrics

Amazon provides with tools to visualize the number of successful deliveries, bounces, complaints and rejects. Such SES metrics can be seen in a bar graph which helps to monitor the effectiveness of notification emails. In addition developers are provided with the tools to handle the bounces, complaints and rejects. As soon as a recipient address marks the email as spam, an email will be sent to the developer about the complaint along with the body of the message sent.

3.3.2 Custom Email with PHPMailer Library

Amazon SES is the SMTP server which acts as the agent to store and deliver the emails. The task of creating the email subject, email body and providing the recipients email address to SES is performed using PHPMailer library.

PHPMailer is an open source php library for sending emails used by about nine million users. It includes implementation of different encryption protocols to prevent eavesdropping of email communications. It is used by many open source projects such as WordPress, Drupal and Yii. Sending high quality emails using php mail function requires a good understanding of SMTP because of the security vulnerabilities such as email injection and spamming. In addition to that HTML email and attachment can be sent using this library. [24]

CASS notification emails were previously sent using Amazon SNS which did not have features to customize the email body. Sending custom emails makes it easier to modify the email body and customize the message, which helps to make the emails genuine.

For sending custom emails using PHPMailer, the library has to be imported in the current version of CASS. A new instance of the PHPMailer class was created that contained functions to update the SMTP server address, username, and password and port number to use are obtained from the Amazon Console for sending the emails. Since two types of emails are sent using CASS, one email per subject to notify about the research she/he is going to participate and notification emails to answer the query, the email type is checked to send the correct email.

As per Amazon's requirement Transport Layer Security (TLS) is enabled for each emails. TLS is the encryption protocol used to secure the communication between a client and server to prevent third-party access or modification to the communication and message [25].

Email addresses of subjects associated with a research are fetched from the database each time a notification emails or subscription emails are sent. The subjects do not have the possibility to unsubscribe the emails. Some email services such as Gmail might treat these emails as spam or malicious. So the official CASS email address should be added to the email contact list. It is necessary to remind the subject to check their spam folder for this reason.

A text file that logs all the successful emails sent is created at CASS for troubleshooting purposes that can arise in the future regarding email notification. The notification email contains a link to the CASS browser client which the subjects can click to directly answer the query. This is useful because a subject does not have to visit the browser client to answer a query. Figure 25 below demonstrates the implementation of the custom email using SES SMTP server and PHPMailer library.

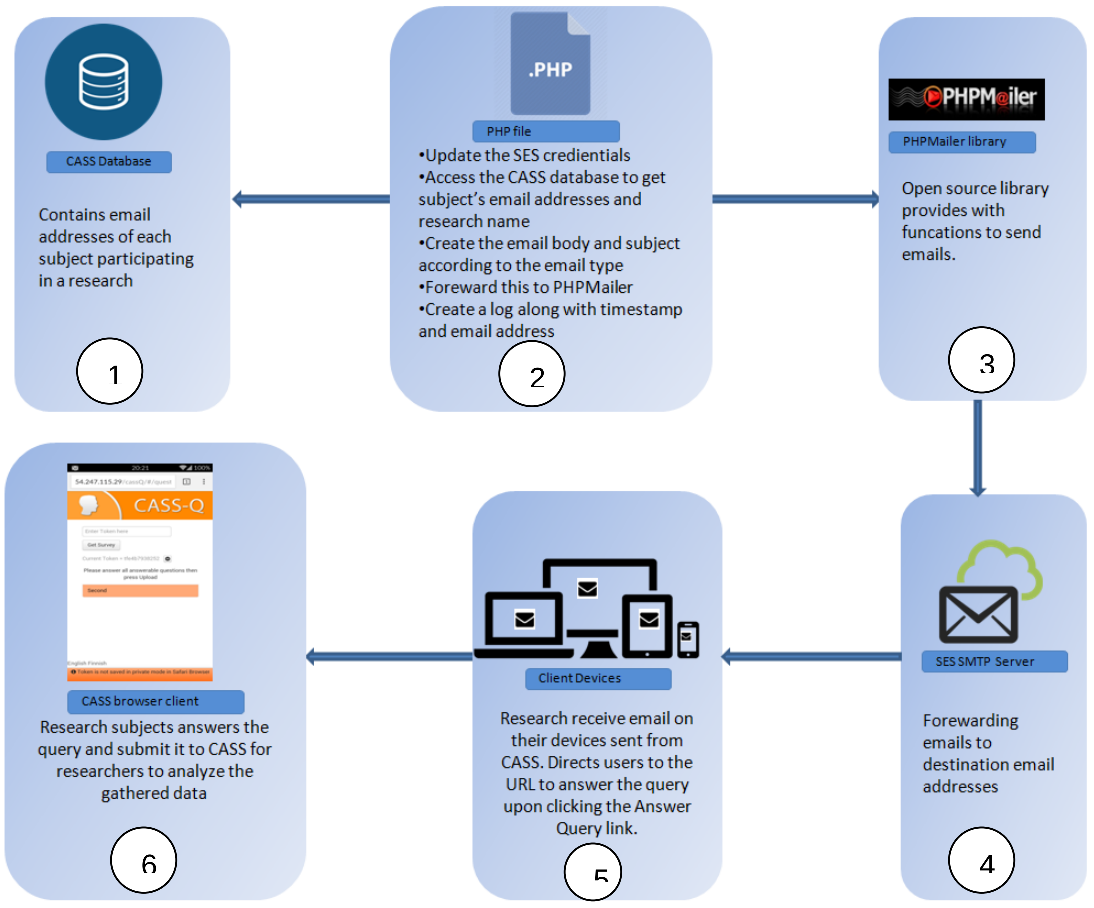
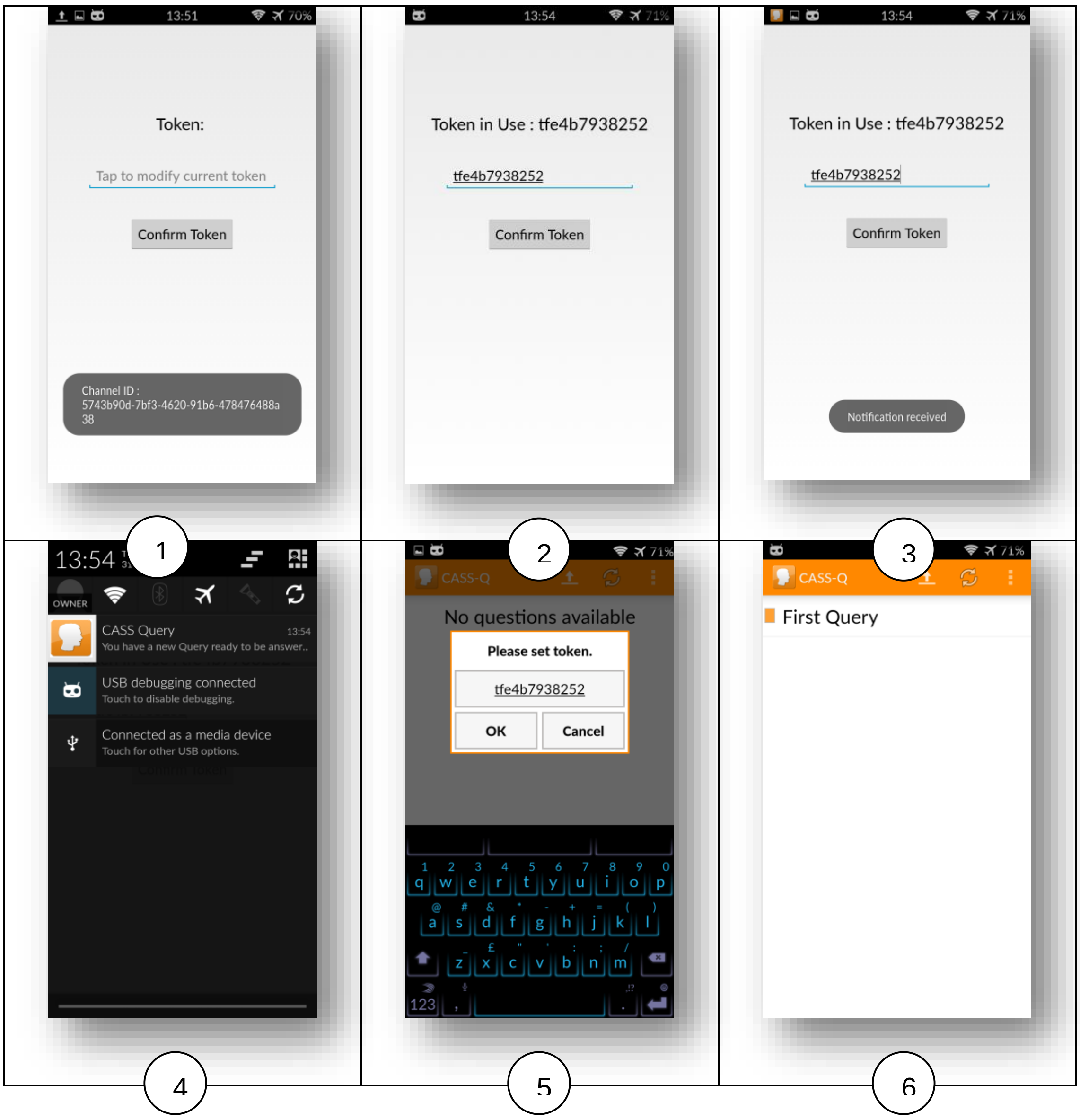

CASS Database
Contains email addresses of each subject participating in a research
1
.PHP
PHP file
•Update the SES credientials
•Access the CASS database to get subject's email addresses and research name
•Create the email body and subject according to the email type
•Foreward this to PHPMailer
•Create a log along with timestamp and email address
2
PHPMailer
PHPMailer library
Open source library provides with funcations to send emails.
3
SES SMTP Server
Forewarding emails to destination email addresses
4
Client Devices
Research receive email on their devices sent from CASS. Directs users to the URL to answer the query upon clicking the Answer Query link.
5
CASS-Q
CASS browser client
Research subjects answers the query and submit it to CASS for researchers to analyze the gathered data
6

Figure 25. Custom email implementation in CASS

As shown in figure 25 steps 1 to 5 involve processes for successful delivery of an email in CASS. Step 6 shows subject using a browser client to answer the CASS query.

4 Results

The results obtained after the implementation of the project meets the previous requirements discussed at the beginning of the thesis project. The goals achieved during this project are illustrated below in this section. The first goal of this project was to implement a cross platform push notification system using third-party notification providers. The use of Urban Airship platform made the push notification service implementation easy to maintain and more transparent to developers for future development. Figure 26 shows the different stages of the CassHelper application developed during this project.



Channel ID : 5743b90d-7bf3-4620-91b6-478476488a38

You have a new Query ready to be answer..

Touch to disable debugging.

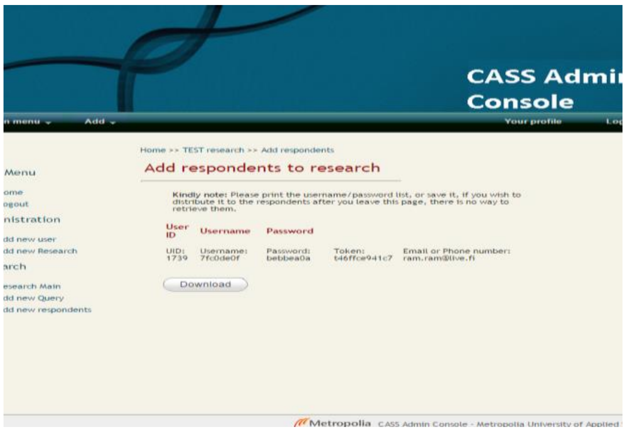
Touch for other USB options.

Figure 26. Different stages of CassHelper application

Figure 26 shows the different stages of CassHelper and CASS-Q application. CassHelper consists of a simple user interface consisting of an input box where user can enter the token associated with the research. A toast shows the channel number created when the application is running for the first time. Once a user enters a token, it will be stored in the application until it is modified or the application is uninstalled. When a notification is received, a CASS logo will be rendered at notification bar, and the phone will vibrate twice and make a system's default notification sound. If the device is in silent or in do not disturb mode, then it will only vibrate twice. When a device is not connected to the Internet, the latest notification is pushed to the device. The task of keeping track of what is the latest notification is done by Urban Airship.

Upon tapping the notification, the user is directed to the CASS-Q application which the research subjects use to answer the queries. If a token is not set in the CASS-Q application, user will be asked to input one when the user is directed to answer the query. Research subjects submit the query via the CASS-Q application previously developed and available in Play Store.

Amazon SNS is no longer used at CASS for notification. Instead the new custom email using Amazon SES is used for email notifications. Two types of emails are sent to a research subject, namely a subscription email and notification email. A subscription email is used to inform the subject about his/her participation in the research. It can include more information about the research. A notification email is sent depending upon the number of the queries the subject has to answer. A notification email is sent as soon as a query is ready to be answered. A log file is used to store the exact time and date when an email is sent. Figure 26 illustrates the research admin creating a new research subject.


CASS Admi
Console

n menu ⌄ Add ⌄ Your profile Lo

Home >> TEST research >> Add respondents

Add respondents to research

Menu
ome
ogout
nistration
dd new user
dd new Research
arch
esearch Main
dd new Query
dd new respondents

Kindly note: Please print the username/password list, or save it, if you wish to distribute it to the respondents after you leave this page, there is no way to retrieve them.

User ID	Username	Password		
UID: 1739	Username: 7fc0de0f	Password: bebbea0a	Token: t46ffce941c7	Email or Phone number: ram.ram@live.fi

Download

Metropolia CASS Admin Console - Metropolia University of Applied

Figure 27. New research subject added by admin

As soon as the new subject is added as shown in figure 27, a subscription email will be sent to the subject. Table 3 and 4 show two different types of emails sent by CASS systems and the differences in look and feel of the emails using Amazon SES and SNS implementation in CASS. Amazon SNS is no longer used in CASS.

Table 3. Two types of emails sent by CASS using Amazon SES.

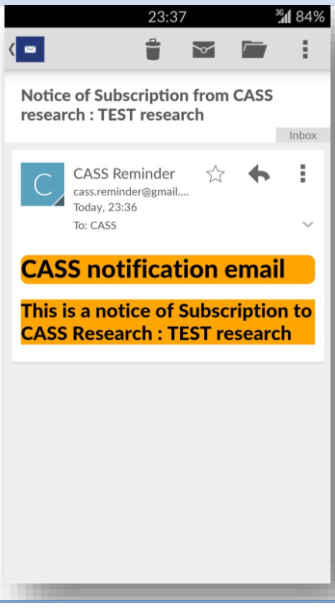
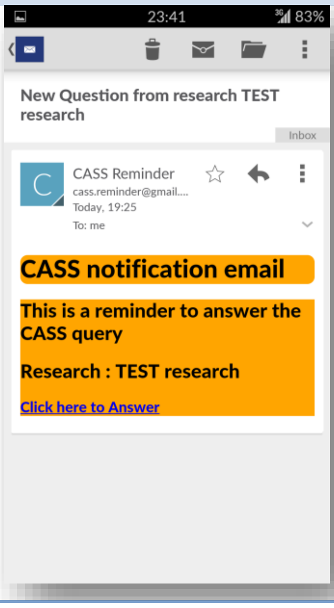
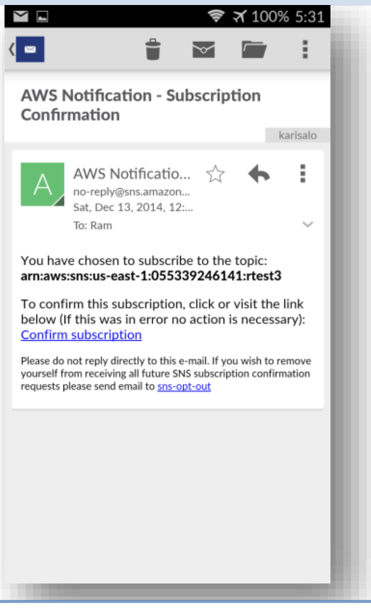
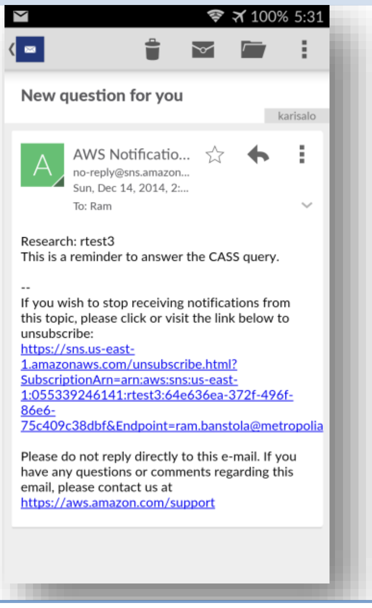
CASS email using Amazon SES	
Subscription email	Notification email
 23:37 84% Notice of Subscription from CASS research : TEST research Inbox CASS Reminder cass.reminder@gmail.... Today, 23:36 To: CASS **CASS notification email** **This is a notice of Subscription to CASS Research : TEST research**	 23:41 83% New Question from research TEST research Inbox CASS Reminder cass.reminder@gmail.... Today, 19:25 To: me **CASS notification email** **This is a reminder to answer the CASS query** **Research : TEST research** Click here to Answer

Table 4. Two types of emails sent by CASS using Amazon SNS (upgraded using Amazon SES).

CASS emails using Amazon SNS	
Subscription Email	Notification Email
	

100% 5:31

AWS Notification - Subscription Confirmation

karisalo

AWS Notificatio...
no-reply@sns.amazon...
Sat, Dec 13, 2014, 12:...
To: Ram

You have chosen to subscribe to the topic:
arn:aws:sns:us-east-1:055339246141:rtest3

To confirm this subscription, click or visit the link below (If this was in error no action is necessary):
Confirm subscription

Please do not reply directly to this e-mail. If you wish to remove yourself from receiving all future SNS subscription confirmation requests please send email to sns-opt-out

100% 5:31

New question for you

karisalo

AWS Notificatio...
no-reply@sns.amazon...
Sun, Dec 14, 2014, 2:...
To: Ram

Research: rtest3
This is a reminder to answer the CASS query.

--
If you wish to stop receiving notifications from this topic, please click or visit the link below to unsubscribe:
https://sns.us-east-1.amazonaws.com/unsubscribe.html?SubscriptionArn=arn:aws:sns:us-east-1:055339246141:rtest3:64e636ea-372f-496f-86e6-75c409c38dbf&Endpoint=ram.banstola@metropolia

Please do not reply directly to this e-mail. If you have any questions or comments regarding this email, please contact us at
https://aws.amazon.com/support

As shown in table 3 and 4, the new email implementation has resulted in a clear email subject and body. If a subject has a problem, it will be also possible to reply the email and get an answer related to it with the new implementation. New emails are more precise and informative for the subjects and developers can easily customize these emails to meet the new requirements of researchers.

The method of how the event of sending an email is triggered, however, remains unchanged. Depending upon the time when the query has to be answered a notification email will be sent to the subject to notify them about the latest query that has be answered as shown in table 4. The research subject can click on the link to directly navigate to the browser client. The results of this email implementation were tested creating a new test research. All the push notifications from Urban Airship and email notification were successfully delivered to the test devices and test email addresses on time. However it should be noted that Gmail treats these emails as spam. So the subject should be advised to check their spam folder if he/she does not receive the email in his/her inbox.

5 Evaluation of the Results

5.1 Benefits

Since CASS is used for collecting data that are context-sensitive data, time plays a vital role in the deviation of data collected. So implementation of an effective notification system plays a vital role in the quality of the data gathered. CASS can be used to study experience, mobile work and professional product design [1]. Email notification for CASS is not effective because it is uncommon for people to check emails frequently. With the growing popularity of smartphones, one can argue that an email can be a good notification medium but it is still not a reliable notification method when a notification has to reach the end user almost in real time. So a push notification is a comparatively better solution for notification when the payload is not big enough. The mobile push notification fits the CASS use case of getting notifications as soon as a query is ready. This new feature adds value to the application by notifying the user immediately and improving the quality of data collected. The analytical data obtained from the Urban Airship console regarding the application usages are beneficial to make the future updates according to the user trend in using the application.

Email implementation can be considered an additional measure to make sure that the notification reaches the research subject. Amazon SNS is a paid service which was implemented at CASS before the new email system. The implementation of the new email system reduces the cost since CASS, being a smaller system, can operate within the free tier with all the benefits of a paid service since CASS is hosted at Amazon Elastic Cloud. Customization of emails is easier using the new custom email. It is more user-friendly and has a better look and feel compared to the Amazon SNS emails. Email statistics obtained from the Amazon console are useful for improving the service as well as understanding if the email is an effective way to send notifications to CASS subjects.

In addition to the benefits discussed above, the project gave me an insight into how testing, debugging and production are carried out in software development projects. I got an opportunity to learn about new technologies and their applications in other projects. Classroom learning are mainly theoretical and general but working to implement a specific feature on system within a given deadline provided work-life experience to me in the technology business. CASS has been under development for a long time. Many features have been added and modified by several programmers who worked with it in the past. So working in this project enhanced my knowledge of understanding the algorithms and logics implemented by other programmers. So overall this project was a boon to me to enhance my career personally and it also added value to the project with new features and improvements.

5.2 Challenges

The project was successful with a few mentionable challenges. The major challenge while implementing these two notification system for CASS was to understand the scheduling procedure of CASS, learning about the Urban Airship library and its implementation to create the CassHelper application and using the PHPMailer library for sending an outbound email. CASS does not have a version control system yet. It came out as a challenge as well. Both Urban Airship and PHPMailer have a well-documented library that consists of example applications and API information. But Urban Airship being a newly used technology does not have a good community for trouble shooting when a problem arises. So before integrating both the notification system into CASS, push notification and email were tested on a small sample applications to make sure that the systems worked.

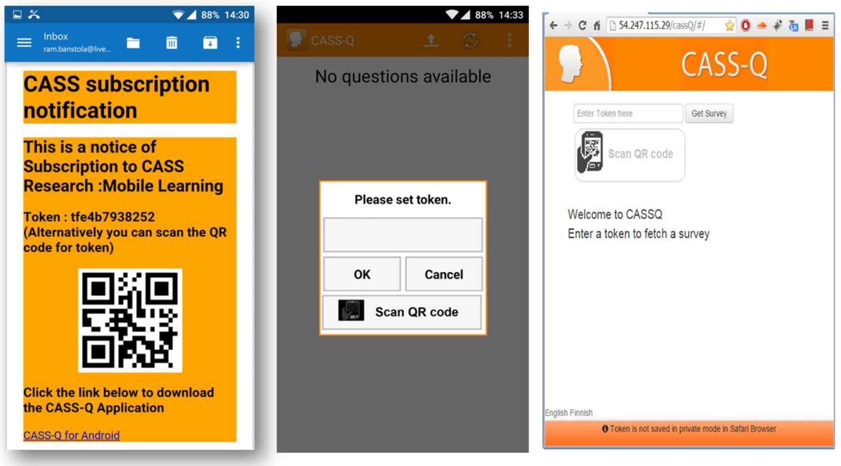
User specification for Urban Airship push notification was a major challenge. There were two options for user specification to utilize the existing token system or create a new token system specific for Urban Airship push. So using the existing token system was decided on. Creating a new token system only for push notification use would be an overhead in the system.

Personally, understanding the programs written by different programmers at different times was a challenge for me. Though CASS has good developer documentation, following which part does which job was difficult in the early phase and it took a considerable amount of time to fix the new bugs that arose at the beginning of the project.

5.3 Future Improvements

Once the project was completed, there were more issues that came into light that could help to make CASS more efficient and user friendly. Prior research on the framework, libraries and notification providers was crucial to the success of this project. So allocating ample time to research for such a task would be useful for the successful implementation of such projects.

However there are a few mentionable future improvements that are doable and would make both CASS backend and client application efficient and user-friendly. With the implementation of custom emails, research notifications and subscription emails can contain more information than just the research name and link to answer the query. The token used in the client application can be distributed via the subscription email. It can include a Quick Response (QR) code that a subject can scan to get the code. There are many open source libraries which can be easily integrated into the existing CASS-Q application to scan QR and to create the QR code on the CASS backend.


88% 14:30

Inbox

ram.banstola@live...

CASS subscription notification

This is a notice of Subscription to CASS Research :Mobile Learning

Token : tfe4b7938252
(Alternatively you can scan the QR code for token)

Click the link below to download the CASS-Q Application

CASS-Q for Android

88% 14:33

CASS-Q

No questions available

Please set token.

OK Cancel

Scan QR code

54.247.115.29/cassQ/#/

CASS-Q

Enter Token here Get Survey

Scan QR code

Welcome to CASSQ
Enter a token to fetch a survey

English Finnish

Token is not saved in private mode in Safari Browser

Suggested CASS subscription email with additional information

Suggested client applications(native and browser) with QR code scan capability

Figure 28. Suggested improvements in CASS email and client applications.

As shown in figure 28, addition of new information such as link to download the client application, token associated with the subject and QR code can make the application more user-friendly. This would reduce the work for researchers who have to give the token manually.

The CASS-Q application is a light-weight application that does not require high performance such as a game application. The browser client is already implemented using AngularJS which makes it easier to implement a cross-platform application using a popular framework such as PhoneGap. This would reduce the complexity to manage a platform specific client application. Push notification using WebSocket can be implemented on the PhoneGap application for each platform.

6 Conclusion

This thesis project was carried out to implement notification systems for CASS. The goals set at the beginning of the project were to implement a new notification email system and a push notification system for CASS. The Amazon SES SMTP server was used for the mail server and PHPMailer library was used for sending email using PHP on the CASS backend. Urban Airship was used for implementing the push notification system. Another notification system discussed in this thesis project was SMS notification and it was discussed as an alternative method and was not feasible at the time of implementation.

The project started with analysing the technologies used and learning about integrating them in the CASS backend and client application. A separate application for receiving push notifications was created because it is easy to test. The push notification system has not been fully integrated into the CASS system even though it is functional but it provides a roadmap for future implementation. Sending emails using an external SMTP server was learnt during the project.

To sum up, the set goals were achieved successfully with new improvements that have to be implemented in the future. The project provided useful knowledge regarding importance of software testing in a test environment and deploying the new features after testing.

References

1. Salo, K., Shakya, U., & Damena, M. (2014). Device agnostic CASS Client. In A. Marcus (Ed.) HCI International 2014: Design, User Experience and Usability 2014, Part II, Lecture Notes in Computer Sciences 8518 (pp. 334-345). Switzerland: Springer International Publishing.

2. Amazon Simple Email Service (SES) [online]. Amazon web services; 28 February 2015. URL: http://aws.amazon.com/ses. Accessed: 5 March 2015.

3. Brain M, Crosby T. How E-mails Works, howstuffworks [online]. URL:http://computer.howstuffworks.com/e-mail-messaging/email.htm. Accessed: 5 March 2015

4. Amazon Simple Email Service Email Sending Best Practices [online].Amazon web services; July 2012. URL:https://media.amazonwebservices.com/AWS_Amazon_SES_Best_Practices.pdf. Accessed : 5 March 2015

5. Government of Senegal boosts Ebola awareness through SMS campaign [online]. Who Health Organization; November 2014. URL: http://www.who.int/features/2014/senegal-ebola-sms/en/ . Accessed: 7 March 2015

6. Rouse M. SMS gateway [online]. TechTarget; July 2012. URL: http://searchmobilecomputing.techtarget.com/definition/SMS-gateway. Accessed: 7 March 2015

7. SMS gateway solution [online].Verma Infotech. URL: http://www.viplworld.com/smsgatewaysolution.aspx Accessed: 7 March 2015

8. Falaki H, Maharajan R, Govindan R. Diversity in Smartphone Usage [online].Microsoft Research; March 2009. URL: http://research.microsoft.com/en-us/um/people/srikanth/data/mobi155-falaki.pdf.

Accessed: 7 March 2015

9. Google Cloud Messaging for Android [online]. Google Inc.
URL: https://developer.Android.com/google/gcm/gcm.html. Accessed: 7 March 2015

10. About Local Notifications and Remote Notifications [online]. Apple Inc.
URL:https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Introduction.html#//apple_ref/doc/uid/TP40008194-CH1-SW. Accessed: 7 March 2015

11. Windows Push Notification Service (WNS) overview (Windows Runtime apps) [online]. URL: https://msdn.microsoft.com/en-us/library/windows/apps/hh913756.aspx. Accessed: 9 March 2015

12. Pocket Guide to Good Push [online. Urban Airship Inc.
URL: http://urbanairship.com/resources/whitepapers/the-pocket-guide-to-good-push. Accessed 12 March 2015

13. Web Notifications [online]. World Wide Web Consortium; 19 January 2015.
URL: https://dvcs.w3.org/hg/notifications/raw-file/tip/Overview.html#introduction. Accessed 12 March 2015

14. Fette I, Melnikov A. The WebSocket Protocol [online]. Internet Engineering Task Force; December 2011.
URL: https://tools.ietf.org/html/rfc6455. Accessed: 12 March 2015.

15. Hanson J. What Are WebSocket? [online]. PubNub;11 September 2013
URL: http://www.pubnub.com/blog/what-are-websockets/. Accessed: 12 March 2015

16. Li T, Zhou X, Xing L, Lee Y, Naveed M, Wang X, Han X. Mayhem in the Push Clouds: Understanding and Mitigating Security Hazards in Mobile Push-Messaging Services. Samsung Research America; November 2014.

17. Constantin L. Cybercriminals use Google Cloud Messaging service to control malware on Android devices [online]. ComputerWorld; 14 August 2013.
URL: http://www.computerworld.com/article/2483740/malware-vulnerabilities/cybercriminals-use-google-cloud-messaging-service-to-control-malware-on-andr.html. Accessed: 12 March 2015

18. Erkkilä J. WebSocket Security Analysis [online]. Aalto University School of Science; autumn 2012.
URL: http://juerkkil.iki.fi/files/writings/websocket2012.pdf. Accessed: 20 March 2015

19. Push Notification Service Comparison [online].b2bCloud; March 2013
URL: http://b2cloud.com.au/reviews/push-notification-service-comparison/. Accessed: 20 March 2015

20. Urban Airship Android Getting Started [online].Urban Airship; January 2015.

URL: http://docs.urbanairship.com/platform/Android.html. Accessed: 20 March 2015

21. What is a cronjob and how do I use it? [online]. FutureQuest Inc; October 2003.
URL:https://service.futurequest.net/index.php?/Knowledgebase/Article/View/23/0/what-is-a-cronjob-and-how-do-i-use-it. Accessed: 20 March 2015

22. Broadcast Receiver [online]. Google Inc.
URL:http://developer.Android.com/reference/Android/content/BroadcastReceiver.html. Accessed: 9 March 2015

23. Amazon SES Product Details [online].Amazon Web Service.
URL: http://aws.amazon.com/ses/details/. Accessed: 20 March 2015

24. Bointin M. PHPMailer - A full-featured email creation and transfer class for [online].
URL :https://github.com/PHPMailer/PHPMailer. Accessed: 20 March 2015

25. Dierks T., Rescorla E.The Transport Layer Security (TLS) protocol Version 1.2 [online].]. Internet Engineering Task Force; August 2008
URL: http://tools.ietf.org/html/rfc5246. Accessed 20 March 2015

Appendix 1: Urban Airship API call from CASS server

```
<?php
// testing details from UA
define('APPKEY', ''); //  App Key from Urban Airship not included shown here
define('PUSHSECRET', ''); // Master Secret
define('PUSHURL', 'https://go.urbanairship.com/api/push/'); // UA API access
class UrbanAirShipPush

{
function __construct()
{
}

/*
*pushUrbanNotification function arguments
*$deviceNumber --> number of CASS subjects
*$tokens --> array of toekn
*$researchName --> name of research
*/
function pushUrbanNotification($deviceNumber, $tokens, $researchName)
{

for ($index = 0; $index < $deviceNumber; $index++)

{

$contents = array();

$contents['alert'] = "You have a new query for research : " . $researchName;

$notification = array();

$notification['Android'] = $contents;

$platform = array();

array_push($platform, "Android");

$token = array();

$token['alias'] = $tokens[$index];

$push = array(


"audience" => $token,


"notification" => $notification,


"device_types" => $platform

);

$json = json_encode($push);

echo "Urban Airship Payload: " . $json;

$session1 = curl_init(PUSHURL);

curl_setopt($session1, CURLOPT_USERPWD, APPKEY . ':' . PUSHSECRET);

curl_setopt($session1, CURLOPT_POST, True);

curl_setopt($session1, CURLOPT_POSTFIELDS, $json);

curl_setopt($session1, CURLOPT_HEADER, False);

curl_setopt($session1, CURLOPT_RETURNTRANSFER, True);

curl_setopt($session1, CURLOPT_HTTPHEADER, array(


'Content-Type:application/json',


'Accept: application/vnd.urbanairship+json; version=3;'

));

$content = curl_exec($session1); //data sent to UA


// echo "Response: " . $content . "\n";
// Check if any error occured

$response = curl_getinfo($session1);

if ($response['http_code'] != 202)

{

echo "Got negative response from server: " . $response['http_code'] . "\n"; // if any error is occured

}

else

{

}

curl_close($session1);
}
}
}
?>
```

Appendix 2: Email via PHPMailer and Amazon SMTP Server

```php
<?php
/**
 *Custom email service for CASS using Amazon's SES(simple email service)
 *@author Ram Krishna Banstola <nep@live.fi>
 *
 */

function send_smtp_mail($mailerlist, $researchName, $typeofemail)
{
                  require_once ('../simpleEmailService/class.phpmailer.php');

 // including library

                  $mail = new PHPMailer(true); // the true param means it will throw exceptions on errors, which we n
eed to catch
                  $mail->IsSMTP(); // telling the class to use SMTP
                  try
                  {
                                    $mail->Host = "email-smtp.eu-west-
1.amazonaws.com"; // SMTP server of Amazon SES. take a notice at
                                    $mail->Port = 587; // SMTP port of the Amazon SES
                                    $mail-
>Username = "##############"; // SMTP username (Amazon SES Credentials)hidden
                                    $mail-
>Password = "##############"; // SMTP password (Amazon SES Credentials)hidden
                                    $mail-
>SMTPDebug = 0; // 1 to enables SMTP debug (for testing), 0 to disable debug (for production)
                                    $mail->SMTPAuth = true; // enable SMTP authentication
                                    $mail->SMTPSecure = "tls"; // tls required for Amazon SES
                                    $mail-
>SetFrom("cass.reminder@gmail.com", "CASS Reminder"); // CASS reminder email
                                    $body = "";
                                    $subject = "";
                                    if (!strcmp($typeofemail, "1"))
                                    {

                  $subject = "Notice of Subscription from CASS research : " . $researchName;
                                                      $body = "<h1 style=' border-radius: 8px; background-
color: orange;  width: 100%; margin-bottom: 5px; '>" . "CASS notification email</h1><div style='background-
color: orange; text-
decoration: none'>" . "<h2 > This is a notice of Subscription to CASS Research : $researchName</h2><h2 style='backgro
und-color: orange'>";
                                    }
                                    else
                                    {
                                                      $subject = "New Question from research " . $researchName;
                                                      $body = "<h1 style=' border-radius: 8px; background-
color: orange;  width: 100%; margin-bottom: 5px; '>" . "CASS notification email</h1><div style='background-
color: orange; text-
decoration: none'>" . "<h2 > This is a reminder to answer the CASS query</h2><h2 style='background-
color: orange'>" . "Research : $researchName</h2><a href='http://54.247.115.29/cassQ/#/questions'><h3>Click here to A
nswer" . "</h3></a></div>";
                                    }

                                    $mail->Subject = $subject;
                                    $mailerlength = sizeof($mailerlist);
                                    for ($index = 0; $index < $mailerlength; $index++)
                                    {

                                                      //  $mail->ClearAllRecipients( ) // clear all

                                                      $mail->AddAddress($mailerlist[$index], "CASS Resarch Subject");
                                                      $mail->MsgHTML($body);
                                                      $mail->Send();
                                                      $mail->ClearAllRecipients();
                                                      cronjoblog($mailerlist[$index]);
                                    }
                  }

                  catch(phpmailerException $e)
                  {
                                    echo $e->errorMessage(); //Pretty error messages from PHPMailer
                  }
                  catch(Exception $e)
                  {
                                    echo $e->getMessage(); //Boring error messages from anything else!
                  }
}

function cronjoblog($emailsent)
{
                  $cronlog = "cronjoblog.txt";
                  $date = new DateTime();
                  $timeStamp = $date->format('Y-m-d H:i:s');
                  $crontext = $timeStamp . "\t\t\t" . "SES email sent to " . $emailsent . "\n";
                  file_put_contents($cronlog, $crontext, FILE_APPEND);
}

?>
```

Appendix 3: Broadcast receiver class in CassHelper Application

```
package fi.metropolia.cass.casshelper;

import com.example.helperapp.R;

import Android.app.NotificationManager;
import Android.app.PendingIntent;
import Android.content.BroadcastReceiver;
import Android.content.Context;
import Android.content.Intent;
import Android.support.v4.app.NotificationCompat;
import Android.util.Log;
import Android.widget.Toast;

public class UrbanBroadcastReceiver extends BroadcastReceiver {

            private int notificationId = 0;

            @Override
            public void onReceive(Context context, Intent intent) {
                        // TODO Auto-generated method stub
                        notificationId++;

                        Toast.makeText(context, "Notification received",
Toast.LENGTH_SHORT).show();
                        Log.i("action-ram", intent.getAction().toString());
                        Intent intent1 = new Intent();
                        intent1.setAction("fi.metropolia.cass.main.CASSI");

                        PendingIntent pIntent =
PendingIntent.getActivity(context, 0, intent1,
                        PendingIntent.FLAG_UPDATE_CURRENT);

                        // Create Notification using
NotificationCompat.Builder
                        NotificationCompat.Builder builder = new
NotificationCompat.Builder(
                        context)
                        // Set Icon
                        .setSmallIcon(R.drawable.cass_icon)
                        // Set Ticker Message

                        // Set Title
                        .setContentTitle("CASS Query")
                        // Set Text
                        .setContentText("You have a new Query ready to be
answered. Tap here to answer!")

                        // Set PendingIntent into Notification
                        .setContentIntent(pIntent)
                        // Dismiss Notification
                        .setAutoCancel(true);
```

```
                                builder.setVibrate(new long[] {
                                            1000, 1000, 1000, 1000, 1000
                                });
                                // Create Notification Manager
                                NotificationManager notificationmanager =
  (NotificationManager) context.getSystemService(Context.NOTIFICATION_SERVICE);
                                // Build Notification with Notification Manager
                                notificationmanager.notify(0, builder.build());

                    }


}
```

Appendix 4: MyApplication class of CassHelper

```
package fi.metropolia.cass.casshelper;

import Android.app.Application;
import Android.support.v4.app.NotificationCompat;

import com.example.helperapp.R;
import com.urbanairship.AirshipConfigOptions;
import com.urbanairship.UAirship;
import com.urbanairship.push.notifications.DefaultNotificationFactory;

public class MyApplication extends Application {

    @Override
    public void onCreate() {
        super.onCreate();



        UAirship.takeOff(this, new UAirship.OnReadyCallback() {
            @Override
            public void onAirshipReady(UAirship airship) {
                // Perform any airship configurations here

                // Create a customized default notification factory
                DefaultNotificationFactory defaultNotificationFactory = new
DefaultNotificationFactory(getApplicationContext());

defaultNotificationFactory.setSmallIconId(R.drawable.ic_notification_button_t
humbs_up);

defaultNotificationFactory.setColor(NotificationCompat.COLOR_DEFAULT);

                // Set it

airship.getPushManager().setNotificationFactory(defaultNotificationFactory);

                // Enable Push
                airship.getPushManager().setPushEnabled(true);
            }
        });
    }
}
```

Appendix 5: MainActivity class of CassHelper

```
package fi.metropolia.cass.casshelper;

import Android.app.Activity;
import Android.content.Context;
import Android.content.SharedPreferences;
import Android.os.Bundle;
import Android.text.Editable;
import Android.text.TextWatcher;
import Android.view.View;
import Android.view.View.OnClickListener;
import Android.view.inputmethod.InputMethodManager;
import Android.widget.Button;
import Android.widget.EditText;
import Android.widget.TextView;
import Android.widget.Toast;

import com.example.helperapp.R;
import com.urbanairship.AirshipConfigOptions;
import com.urbanairship.UAirship;

public class MainActivity extends Activity {

            private EditText tokenText;
            private TextView tokenTextView;
            @Override
            public void onCreate(Bundle savedInstanceState) {
                        super.onCreate(savedInstanceState);
                        getActionBar().hide();
                        setContentView(R.layout.activity_main);
                        Button buttonConfirm = (Button)
findViewById(R.id.buttonConfirm);
                        buttonConfirm.setOnClickListener(new
OnClickListener() {

                                    @Override
                                    public void onClick(View v) {
                                                // TODO Auto-generated method
stub
                                                InputMethodManager
inputManager = (InputMethodManager)
getSystemService(Context.INPUT_METHOD_SERVICE);


            inputManager.hideSoftInputFromWindow(getCurrentFocus()

            .getWindowToken(), InputMethodManager.HIDE_NOT_ALWAYS);

                                    }
                        });

                        final AirshipConfigOptions options =
AirshipConfigOptions
                                                .loadDefaultOptions(this);
                        // options.developmentAppKey =
"tdkXr7lBTnuLPJNd0wqqqg";
                        options.inProduction = false;
                        // Enable Push
                        tokenText = (EditText)
findViewById(R.id.tokenTextEdit);
                        tokenTextView = (TextView)
findViewById(R.id.textView1);

                        SharedPreferences sharedPref = this

            .getPreferences(Context.MODE_PRIVATE);
                        final SharedPreferences.Editor editor =
sharedPref.edit();

                        if (!sharedPref.getBoolean("runstat", false)) {
                                    Toast.makeText(getApplicationContext(),
"First time use",

            Toast.LENGTH_SHORT).show();

                                    editor.putBoolean("runstat", true);
                                    editor.commit();

                        } else {
                                    Toast.makeText(

            getApplicationContext(),
                                                            "Channel ID : \n"

            + UAirship.shared().getPushManager().getChannelId(),

            Toast.LENGTH_LONG).show();
                                    String token =
sharedPref.getString("token", "");

                                    tokenTextView.setText("Token: " + token);
                                    tokenText.setHint("Tap to modify current
token");

                        }

                        tokenText.addTextChangedListener(new TextWatcher() {

                                    @Override
                                    public void onTextChanged(CharSequence s,
int start, int before,
                                                            int count) {
                                                // TODO Auto-generated method
stub

                                    }

                                    @Override
                                    public void
beforeTextChanged(CharSequence s, int start, int count,
                                                            int after) {
                                                // TODO Auto-generated method
stub

                                    }
                                    @Override
                                    public void afterTextChanged(Editable s)
{

                                                String token =
tokenText.getText().toString();
                                                tokenTextView.setText("Token
in Use : " + token);

            UAirship.shared().getPushManager().setAlias(token);
                                                editor.putString("token",
token);
                                                editor.commit();
                                    }
                        });

            }

}
```