Research on Social Context Acquisition and Reasoning Techniques for Online Social Networks

Baraa Bakhsh

A THESIS SUBMITTED TO AUCKLAND UNIVERSITY OF TECHNOLOGY
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF COMPUTER AND INFORMATION SCIENCES

School of Engineering, Computer and Mathematical Sciences

March 2017

Copyright

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the qualification of any other degree or diploma of a university or other institution of higher learning.

Signature of candidate

Acknowledgements

First and foremost, I thank almighty Allah the most gracious and the most merciful, for giving me the opportunity, strength, and determination to complete my thesis. This thesis was achieved at the Faculty of Design and Creative Technologies in the School of Engineering, Computer and Mathematical Sciences at Auckland University of Technology, New Zealand.

Second, I would like to thank my mum: Dr.Najat Bantan, and my father: Fayez Bakhsh for their limitless support over the past few years in my journey of studying in New Zealand. I would also like to express my deepest appreciation to my wife: Lana Kotby for her continuous support and courage during my study, for being cheerful whenever I had stressful times, and for understanding whenever I could not spend quality time with her. She also helped me a lot in finalizing this thesis within the limited time frame.

Third, I would like to express my special thanks of gratitude to my supervisor: Dr.Jian Yu who gave me the golden opportunity to do this wonderful project on the topic: Social Context Acquisition and Reasoning Techniques for Online Social Networks, which also helped me in doing a lot of Research and I came to know about so many new things. I am really thankful to him.

Finally, I would like to thank my friend: Emad Alsaiari for sharing ideas with me that helped throughout my thesis. I would also like to thank the Saudi Arabian cultural Mission for facilitating the process of studying in a foreign country.

Abstract

Recently, social contexts have been used with promising outcomes in developing social aware applications in different fields. However, a user's social information is distributed over different Online Social Networks (OSNs), which is a challenge for developers wishing to collect this information. Integrating social contexts from these resources can provide such useful applications. In this thesis, a Social Context Web Service (SCWS) framework architecture is being proposed. This framework has the ability to acquire raw social contexts from Facebook; an ontology-based model designed for classifying, inferring, and storing social contexts, as a proof-of-concept. In addition, CommonFriends application has been developed that connects with the framework through Application Programming Interfaces (APIs) to notify users when any of the following scenarios happen: finding common friends, notifying the user of their distance from their friends, when they are within the same location, and suggesting a new friend with common interests. These features will demonstrates the applicability of the research approach. The performance of the mobile application has been evaluated using tester accounts under Facebook's testing environment.

Contents

List of Tables

List of Figures

List of Glossaries

API Application Programming Interface

BDR Battery Depletion Rate

CML Context Modelling Language

CPU Central Processing Unit

DDS Data Distribution Service

DHT Distributed Hash Table

DSRM Design Science Research Methodology

FOAF Friend-of-a-Friend

GPS Global Positioning System

HiBOp History Based Routing Protocol for Opportunistic Networks

HTTP Hypertext Transfer Protocol

IBI Interactive Battery Interface

IDE Integrated Development Environment

IntEO Interaction Event Ontology

IP Internet Protocol

JSON JavaScript Object Notation

M-Commerce Mobile-Commerce

mA Milliampere

MAC address Media Access Control Address

mW Milliwatt

OAuth Open Authorization

OS Operating System

OSN Online Social Network

OWL Ontology Web Language

P2P Peer-to-Peer

PSN Pervasive Social Network

QTP Qualcomm's Trepn Profiler

RDF Resource Description Framework

RDFS Resource Description Framework Schema

RSSI Received Signal Strength Indication

RUDP Reliable UDP Protocol

SA Social Acquisition

SC Social Context

SCM Social Context Modelling

SCOnto Social Context Ontology

SCWS Social Context Web Service

SDDL Scalable Data Distribution Layer

SDK Software Development Kit

SOAP Simple Object Protocol

SP Social Provision

SPARQL SPARQL Protocol and RDF Query Language

SQL Structured Query Language

SSR Social Situation Recognition

URI Uniform Resource Identifier

URL Uniform Resource Locator

VM Virtual Machine

W_{app} Weight of the Application

XML eXtensible Markup Language

Chapter 1

Introduction

1.1 Background and Motivation

Context-awareness has become a major topic in many different areas of computer science; for example, pervasive and ubiquitous computing (Henricksen & Indulska, 2006; Serral, Valderas & Pelechano, 2010), Human Computer Interaction (Flentge, Weber, Behring & Ziegert, 2008) and wireless sensor network (Wood et al., 2008; Gámez, Cubo, Fuentes & Pimentel, 2012). The terminology *context-awareness* was first introduced by B. N. Schilit and Theimer (1994) and can be defined as pervasive computing systems that alter their operations to the current situation without the intervention of the users. Hence, systems can be more intelligent and effective to deliver a useful information for users by analysing their contextual data (Baldauf, Dustdar & Rosenberg, 2007; Dey, 2001).

Context-awareness is a combination of two aspects: sensor technology, and software engineering technology. Sensor technology is the contextual information which is captured from complex (physical and social) environment from a network, using sensors such as GPS, microphone, camera(Gámez et al., 2012) from a local device, while, software engineering technology is related to interpreting, and processing sensor

information (Dey, Abowd & Salber, 2001; Gu, Pung & Zhang, 2004), managing and storing contextual data (Gu et al., 2004) addressing security, and privacy issue (Kulkarni & Tripathi, 2008; Zhang & Parashar, 2004) and providing the data to application solutions (Van Kranenburg, Bargh, Iacob & Peddemors, 2006) a social, with context-aware, application.

In the last decade, context-awareness has been implemented in different fields in application such as tour guides (Cheverst, Davies, Mitchell, Friday & Efstratiou, 2000; Park, Hwang, Kim & Chang, 2007), Mobile-Commerce (M-Commerce) (Drakatos, Pissinou, Makki & Douligeris, 2007), learning systems (Chen & Li, 2010), and mobile computing (Beach et al., 2010). At the same time, context-aware services have been considered as an important tool, particularly in the smart city. This field has been a large concern for many organizations, such as IBM, Intel and Siemens because each of them are competing to take the lead (Byun & Park, 2011; Solanas et al., 2014).

Despite the amount of research that has been achieved in this field, the development of OSN; for example, Facebook, LinkedIn and Twitter is surpassing expectations. As of January 2016, the most well-known OSN site, Facebook has exceeded 1 billion registered users, 1.55 billion of whom are active per month (Statista, 2016). This indicates that OSN contains huge volumes of social context information; for instance, name, picture, friend lists, favourite books, movies. Thus, this contextual information can be used to build unique applications (Beach et al., 2008).

Social context is the basis of accomplishing social-awareness. There are various researches that define what social context is being considered, but the description of all of them is based on their needs and situations. Social Context refers to the kinds of data obtained from relationships and interactions with surrounding people (Han, Jyri, Ma & Yu, 2008; Eugster, Garbinato & Holzer, 2009; Zheng & Yano, 2007; G. Wang, Gallagher, Luo & Forsyth, 2010). Social context only happens between group of people and it can be divided into three classes: actor class, relation class, and event class.

- Actor class context is the information that characterize an individual's personality, such as name, age and gender.
- Relation class context refers to the relationship information between two or more characters, such as families, colleagues, friends, players.
- Event class context describe the information related to events which are organized and attended by people.

  Event class context also can be classified into two sub-classes:

  - Actor event context is formed by individual activities.
  - Interaction event context is identified from user's interaction activities.

The location of the users can be categorized into three classes based on their situation:

- if the location is the user's home address, then the social context belongs to "social profile".
- if the location is a place that the user visited, then the social context belongs to "individual event".
- if the location is a place of a football match, then the social context belongs to "interaction event".

Improving the behaviour and the humanization of the application in a situation can be only achieved by utilizing social context information, so context aware applications that consider analysing social context is more suitable; thus, if the relationship between two users is identified, the application would provide some useful notification.

However, the challenges of developing social-aware infrastructure remain; therefore, the objective of this paper is to propose a context-aware middleware architecture SCWS, which aims to build a comprehensive management system. Because this framework collects raw social data from multiple OSNs, another aim is to design an ontology in order to represent contextual knowledge about the users and their social situations.

Finally, it provides support for cross-platform interfaces that allow applications to use the contextual data.

1.2 Research Questions

In recent years, because of the rapid growth of OSN, a few social-aware applications have emerged. However, there are still significant challenges that remain to be investigated in order to efficiently develop and operate social context-aware web services, in particular in context modelling and reasoning and data acquisition from various sources. The aim of this paper is to answer the following questions:

- How to model social context. Until now, there has been no commonly agreed definition and conceptual model for social context (Kabir, Han, Yu & Colman, 2014).
- How to effectively acquire social context information from various sources like OSNs including Facebook, LinkedIn, and Twitter.
- How to populate the social context model with actual context data.
- How to manage the social context model and social context information so that it can be effectively used by software applications and services.
- How to rapidly develop such software services so it can dynamically adapt to changes to the user's social context (J. Yu et al., 2015).

1.3 Contribution

SCWS is an architecture that aims to enhance the maagement framework of the social contexts. Overall, the main goal of this framework is intended to make the following contribution:

- A framework architecture is proposed to enhance the service of social awareness by using social contexts from different OSNs.
- Ontology Web Language (OWL)-based: a comprehensive ontology is designed to accommodate social contexts.
- A technique is suggested to populate social contexts into the designed ontology-based model.
- A web service APIs is developed to enable applications in cross platforms to use these contexts.
- A recommendation is provided to help future researchers to decide on what to use in order to build social awareness applications more quickly.

1.4 Research Structure

This research is organized into five chapters; Chapter 2 reviews several related studies, the meaning of contexts and social contexts, and how most studies' users have defined them. It also discusses the current state of the frameworks in terms of the technologies used for communications: Bluetooth, and Wireless; the architecture: Peer-to-Peer (P2P), centralized, or semi-centralized; the type of contexts: Global Positioning System (GPS), or OSNs; and the model: OWL.

Chapter 3 shows the methodology used in this study. In addition, it describes the design of the research, and the stages that the researcher has performed. It also demonstrates the proposed architecture, and the concept of it. Later, it explains how the ontology has been designed, and its contents. Then, it illustrates the environment of the laboratory and the tools used. Finally, it determines how the data are generated, what data are collected, and how the data are presented.

Chapter 4 presents the findings of the approach. These findings are explained as scenario cases to demonstrate the applicability of the approach. Then, they are presented

in a graphical manner. Later, lab experiments are implemented to evaluate the final artifact in terms of the performance, and the results are represented in charts.

Chapter 5 provides an answer for the research questions presented in section 1.2. It also compares the solution to previous frameworks proposed in previous studies. Likewise, it discusses the results obtained from the lab experiment that evaluated the final product. Lastly, it presents the features and the missing features of the tools used in the research.

Chapter 6 summarises the thesis' main findings. It also provides a recommendation for best practice. Additionally, it addresses the limitations of this research, and indicates future work.

Chapter 2

Literature Review

2.1 Introduction

In this chapter, the existing functional requirements of social context middleware will be discussed. To provide an efficient social context-awareness; the middleware should support inference services of different social contexts. Moreover, the middleware should include context acquisition, context model, context reasoning and social tie inference.

This Chapter is organized into seven sections that provides an extended summary of relevant works, and specifies the potential direction for this research. Section 2.2 defines context and describes the social context. Section 2.3 explains the available techniques for acquiring data from OSNs. Section 2.4 describes different types of radio channels, and protocols. Section 2.5 discusses various context models, and identifies their weaknesses and strengths. Following that, Section 2.6 shows different reasoning methods, and it presents the advantages and disadvantages of each method. Section 2.7 illustrates briefly about social tie inference, and how it has been used. Section 2.8 demonstrates different methods for achieving a secure middleware. Finally, Section 2.9 reviews the most significant aspects of the types of middleware that integrate OSNs.

2.2 Context

The meaning of context varies, and it has been defined in many studies according to the user. For example, B. N. Schilit and Theimer (1994) define context as locations, identities of surrounding people, and objects, while Ryan, Pascoe and Morse (1999) state context as the location of the user, environment, identity, and time. However, one of the most comprehensive definitions is given by Dey and Abowd (1999) as:

> any information that can be used to characterize the situation of entities (i.e., whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves . (p. 3).

All of these definitions are related to what the researchers were developing or building. The most-used contextual information was the user's location (Baldauf et al., 2007). When Want, Hopper, Falcao and Gibbons (1992) built the first context-aware application called *Active Badge Location System*, this was used to forward phone calls to a telephone near the user depending on their location. Another study created tour guide applications based on the current location of the user to provide travellers with useful information, for instance, the application could suggest things to do or places to visit on the map (Abowd et al., 1997; Sumi et al., 1998; Cheverst et al., 2000).

Recently, context has been broken down into three types: Firstly, context relates to a service being requested, such as the identification information of the requester, personal preferences, current situation, and friend lists. Secondly, context relates to a Web service, including service location, service status, and its quality of service attributes. Finally, context relates to time and weather information. In addition, some contextual information is received directly from physical sensors; namely, locations and temperatures. Other contexts can be extracted from a context provider, like an OSN (Sheng & Benatallah, 2005). Having defined what is meant by context, the next section of this paper addresses social context.

2.2.1 Social Context

Social context is defined as any piece of information that is shared on the online society that describes a person's social information. This information can be obtained from devices, network providers and OSNs in order to improve the ability of social context-awareness. Social information can be profiles, interests, preferences, behaviours, social relations, social activities, digital content, and the geographical environment surrounding the person (Kabir, 2013; Xing, Gronowski, Radia, Svensson & Ton, 2011).

The above definition only covers the public information of the users, excluding private information. Considering the meaning from Oxford Dictionary the term *social* is described as a society or an organization (Social, 2016). In that environment, individuals can interact and exchange information with other individuals.

Social context can be categorised into object-centric relationships and people-centric relationships. Object-centric relationships contain information of people who have a common interest, take a part in common activities, or join similar groups (Kabir, Han, Yu & Colman, 2012). These relationships can benefit applications by inferring a user's preferences (Mislove, Gummadi & Druschel, 2006) and encouraging the sharing of resources (Li & Dabek, 2006). On the other hand, people-centric relationships are formal and explicit; for example, if an individual identifies other individuals as father, boss or classmates. These relationships can be used in social-aware phone applications (Toninelli, Khushraj, Lassila & Montanari, 2008).

2.3 Context Acquisition

There are two techniques for context acquisition: query-based and event-driven. Conducting query-based mechanism enables applications to send queries to servers or

sensors to retrieve the desired contexts. In addition, it offers a Structured Query Language (SQL)-like interface for query function. In contrast to query-based, event-driven mechanism splits the event into two steps: the first is to define the event that indicates certain state changes of the data in order to detect the event; the second step is that notification of an event is sent by the middleware to the applications that are interested in that event (Liang & Cao, 2015).

SCIMS uses a generic query interface to import social context from OSNs (Kabir et al., 2012). In Yarta also, the middleware conducts remote queries to retrieve social data from other nodes. Heterogeneous network interfaces, and connecting technologies are used to transfer the data (Toninelli, Pathak & Issarny, 2011). In Whozthat, a mobile device sends a query to acquire context data of another device from an OSN to predict the relationship between users. The protocol in Whozthat also provides support to heterogeneous links and OSNs (Beach et al., 2008).

Context acquisition also needs to retrieve information from local contexts. In MobiClique, when two devices are near one another, data such as friendship and interests are exchanged over Bluetooth. MobileClique uses an event-driven mechanism as its core system. It comprises of a single queue event and a group of managers that detect and respond to different system events, including finding new neighbours, and inbound contexts from local applications. MobiClique adopts Haggle architecture (Pietiläinen, Oliver, LeBrun, Varghese & Diot, 2009). The kernel of Haggle is a single event queue that starts processing the messages after being stored (Boldrini, Conti, Delmastro & Passarella, 2010); thus, the time required for a request depends on the status of the queue. This section has analysed various techniques of context acquisitions, the next part of this paper discusses the meaning of OSNs.

2.3.1 Online Social Networks (OSNs)

Online Social Network (OSN) is a virtual world (Kaplan & Haenlein, 2010) that enables people to create a public or a semi-public profile to connect with other users within the system (Ellison et al., 2007). According to the Cambridge Dictionary, Social Network (2016) is an application that enables an individual to communicate and share information with another on the internet. The primary purpose of OSN is not to meet a stranger but to communicate with the existing friends by sharing shared contents: interests, activities, and status messages for triggering communication (Haythornthwaite, 2005). One of the largest OSNs is Facebook that allows users to communicate by providing every users with *Live Feeds* that streams other user activities (Levy, 2010). In addition, LinkedIn allows workers and employers to create a professional profile in order to find jobs, join a company discussion to share thought (Kietzmann, Hermkens, McCarthy & Silvestre, 2011). It is noticeable that the main contents of OSNs rely on sharing among others. These contents are considered as social contexts; according to the definition mentioned in Section 2.2.1; therefore, users and groups are the primary factor of social contexts, as shown in Figure 2.1.

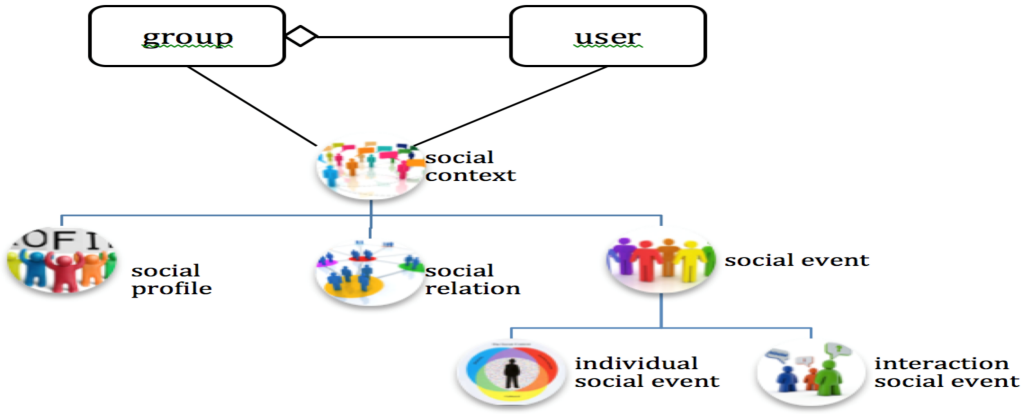


Figure 2.1: Users and Groups in Online Social Networks (OSNs)

OSNs provide a tool that allows for access to social information and allows third-party developers to include them into their application through an API (Facebook, 2016; LinkedIn, 2016a). Query API is a manner of acquiring social contexts from a server using simple Hypertext Transfer Protocol (HTTP) requests (Masse, 2011).
The following section describes OSNs that offer APIs: Facebook and LinkedIn.

2.3.1.1 Facebook

Facebook is one of the most preferred OSNs (Duggan & Page, 2015). According to Facebook developer website (2016), Facebook API is called Graph API providing a chain of APIs that allow developers to retrieve social information, but these developers are required first to obtain authorization from Facebook. This Graph holds all the information within the Facebook's platform and it is connected with all the user's accounts. It is a low-level HTTP-based API that enables the developers to perform varieties of tasks. Facebook contains a huge amount of context information, such as people, photos, events, pages, and comments. The following Figure 2.2 shows the data model of Facebook.

Despite the fact that Facebook offers an API for third-party application, social information cannot be acquired automatically. Facebook only permits a social context that the users set as public.

Facebook also provides numerous types of technologies that allows different programming languages to integrate Facebook into new applications, such as PHP Software Development Kit (SDK), and JavaScript SDK for web development, and Android SDK, and IOS SDK for mobile development.

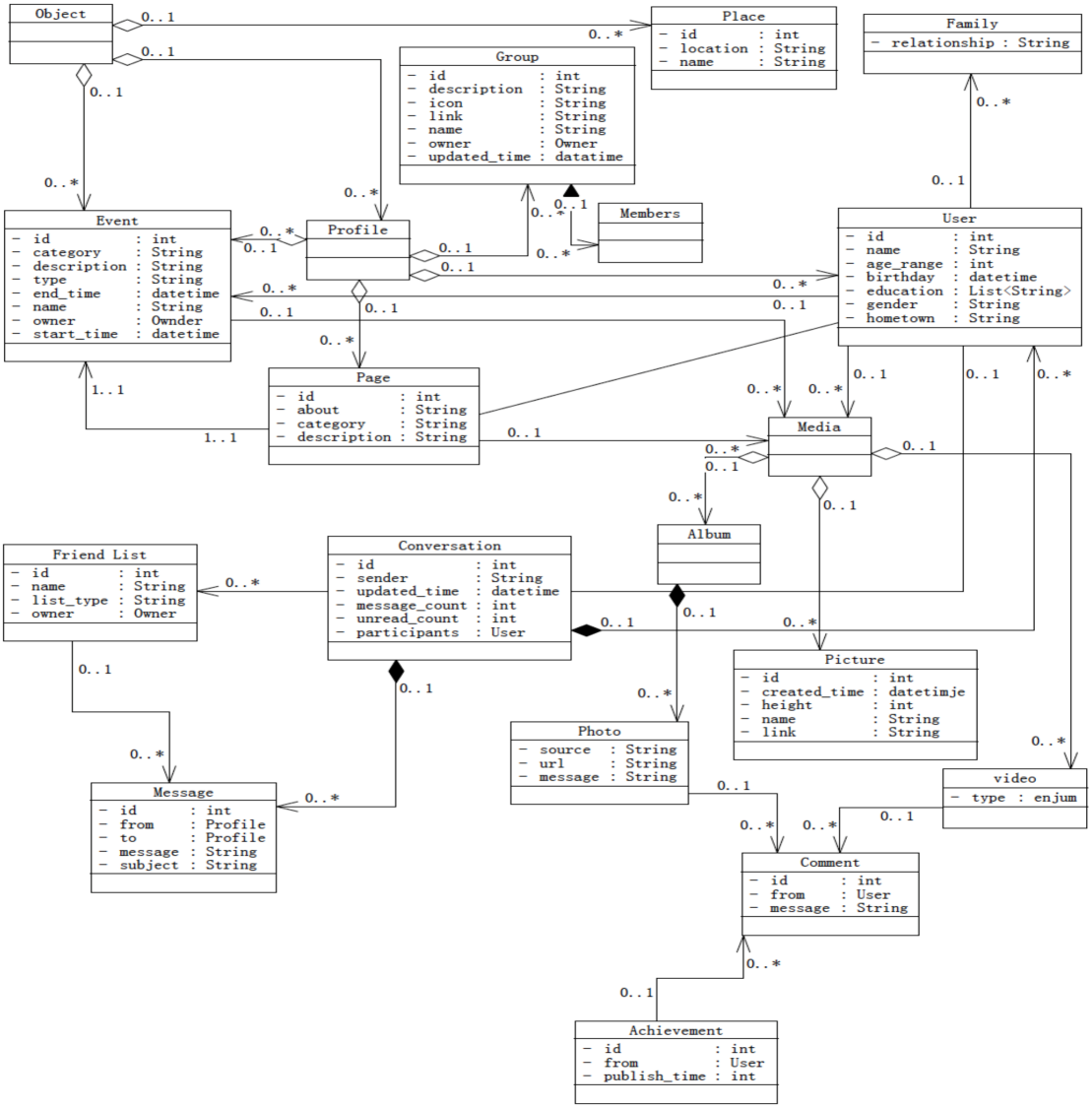


Figure 2.2: Facebook Model

2.3.1.2 LinkedIn

LinkedIn is the largest business social network in the world. Since May 2003, almost 400 million users from over 200 countries worldwide have joint the network (LinkedIn, 2016b). LinkedIn provides a service that enables users to keep a list of people who they work or have business together, which is called connections. Each member of the service provides two groups of contexts: people and company, as shown in Figure 2.3.

LinkedIn also offers different kinds of technologies, such as JavaScript, Mobile SDK for Android and IOS wrapper via the default API. This API supports both eXtensible Markup Language (XML), and JavaScript Object Notation (JSON) response formats.

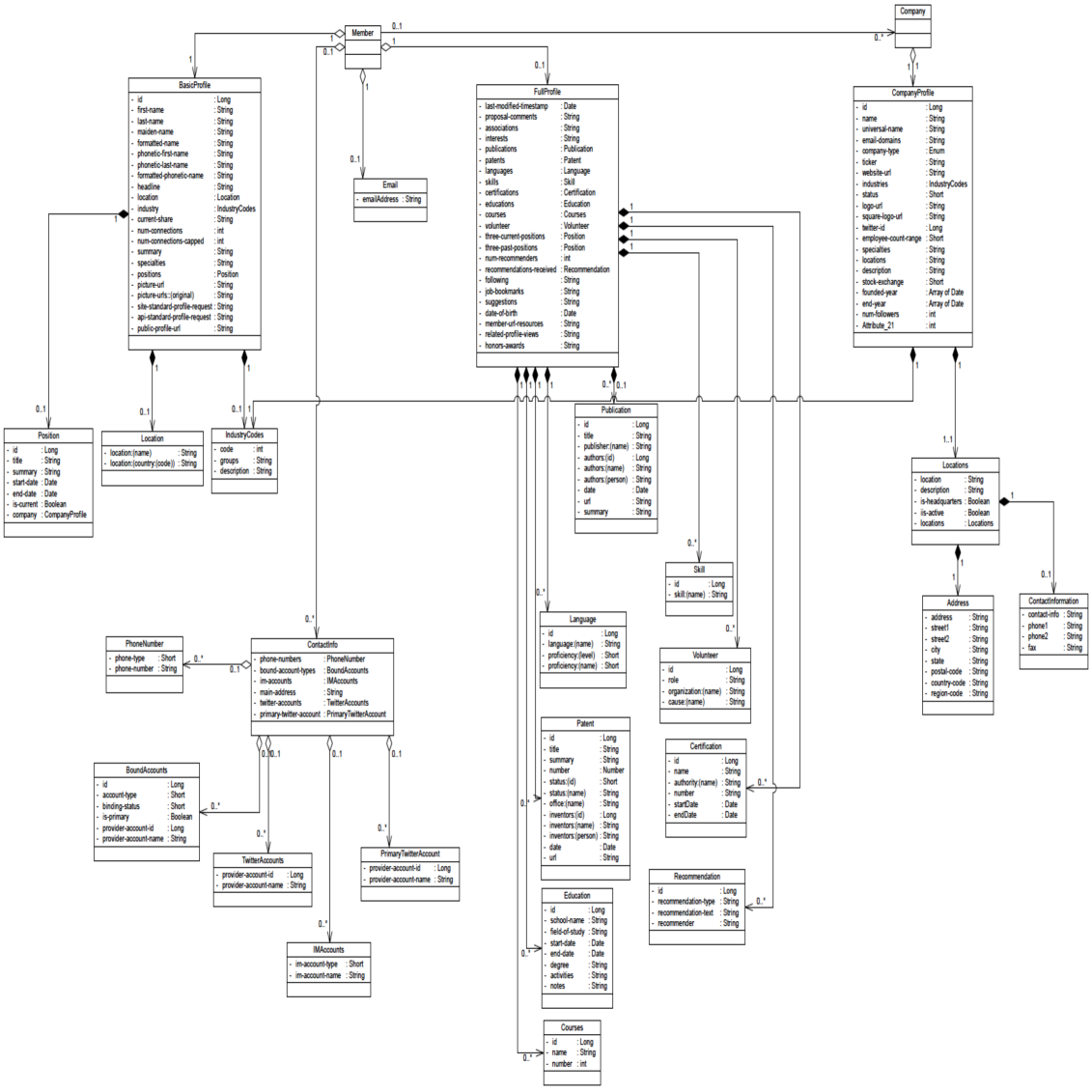


Figure 2.3: LinkedIn Model

2.4 Communication Channel

There are several radio channels that allow different devices to communicate among each other, such as Bluetooth, and Wi-Fi. Wi-Fi or cellular telephone are the most suitable techniques for mobile devices to communicate with servers because they are deployed on a large scale (Pietiläinen et al., 2009).

There are various studies of social context-awareness that have implemented different networking architectures. These architectures are described as follows:

P2P is the most applied architecture in the proposed middleware. It enables the users to manage their privacy, but it consumes the Central Processing Unit (CPU) of the device and storage (Liang & Cao, 2015).

Example of Middleware: CAMEO (Arnaboldi, Conti & Delmastro, 2014), MobiClique (Pietiläinen et al., 2009), Prometheus (Kourtellis et al., 2010), VENETA (Von Arb, Bader, Kuhn & Wattenhofer, 2008), Whozthat (Beach et al., 2008) and Yarta (Toninelli et al., 2011).

Centerlised relies on a central server where all processes are performed; for example managing and storing social contexts which are then sent to the client. This generates some problems with privacy (Liang & Cao, 2015).

Example of Middleware: CircleSense (Liang, Cao & Zhu, 2013), MobiSoC (Gupta, Kalra, Boston & Borcea, 2009) and SCIMS (Kabir et al., 2012).

Semi-Distributed defines some criteria for which a device will act as a server. For instance, only mobiles with the highest activities are elected to be a broker. This broker exchanges data between devices and infers missing social links. Therefore, a broker can find a link between users to provide precise recommendations (Mokhtar, McNamara & Capra, 2009).

Example of Middleware: Pervasive Social Network (PSN)

On top of the above networking architectures, there are different network protocols

that are responsible for delivering messages. Firstly, in the centralized architecture, network protocol controls the communication among mobile devices and the server. According to MobileHealthNet developed by Teles et al. (2012), the communication infrastructure of the middleware is built on a Scalable Data Distribution Layer (SDDL). SDDL is able to implement two types of communication protocols: Data Distribution Service (DDS), and Reliable UDP Protocol (RUDP). DDS is applied for the wired communication within the application's network, while RUDP is implemented for the incoming and outgoing communication between the core of the network and the mobile nodes. RUDP provides a publish/subscribe method with a unicast communication, which means a single sender and receiver between the mobile nodes and the gateway. It also can deal with the recurrence of the disconnectivity. But MobiSoc has selected a protocol based on Simple Object Protocol (SOAP). This protocol uses XML information as a format and HTTP for communication between the server and the users. SOAP has two advantages: 1) it is an independent language. 2) it is supported by several popular languages. However, due to the redundancy of the protocol and the frequent update of social context, SOAP is not the most suitable protocol (Gupta et al., 2009).

Secondly, within P2P architecture; there are different routing and forwarding protocols for opportunistic networking (Pelusi, Passarella & Conti, 2006). CAMEO embraces a History Based Routing Protocol for Opportunistic Networks (HiBOp) forwarding protocols for delivering messages to multi-hop destinations (Arnaboldi et al., 2014). In HiBOp, messages are only delivered to nodes that are likely to reach the target (Boldrini, Conti, Jacopini & Passarella, 2007). In MobiClique, the forwarding protocols depend on two simple rules: The first is that messages are sent either directly to the specified user or through a friend of the user. The second rule is that a group of messages are received to the corresponded group, till every user of the group receives the data (Pietiläinen et al., 2009). VENETA is based on a Bluetooth messaging protocol that can forward messages up to three hops through relating routing (Von Arb et al., 2008). Prometheus

uses Pastry which is established on the top of a Distributed Hash Table (DHT)-based overlay (Kourtellis et al., 2010). Pastry delivers messages with a key to nodes whose node identifier is near the key numerically. (Rowstron & Druschel, 2001).

SAMOA adopts a UDP-based protocol where both point-to-point and multi-point communication can be used. Point-to-point communication allows entities to deliver messages to a destination with a known Internet Protocol (IP), whereas multi-point communication enables entities to send messages to different entities within the same place (Bottazzi, Montanari & Toninelli, 2007). Whozthat employs both Bluetooth among devices and Wi-Fi for the communication between the devices and the OSNs (Beach et al., 2008). Yarta applies iBlCOOP middleware (Toninelli et al., 2011) that allows devices to communicate over different networks(Bennaceur, Singh, Raverdy & Issarny, 2009). The communication manager in Yarta can use both synchronous and asynchronous message transmitter through multi-radio links. It also can cope with intermittent connectivity and the data are transmitted over heterogeneous network (Toninelli et al., 2011).

2.5 Context Model

Developing a context-aware system is not an easy task and consumes a much time because there have been no appropriate infrastructures that can handle acquiring context from various sources, such as physical sensors, databases, and agents; performing context interpretation; carrying out dissemination of context to interested parties in a distributed and timely fashion; and providing programming models for constructing of context-aware services. To provide a convenient context-aware application, a well designed context model should be established (Gu et al., 2004).

There are different types of methods that have been proposed for modelling contexts.

The following section reviews some of these models: key-value, object-role, ontology-based and hybrid.

2.5.1 Key-Value

Key-Value model utilizes key-value pairs to demonstrate different data (Liang & Cao, 2015). This model is not complex for data representation, and it can only be performed on a simple social profile, such as a contact list, despite the fact that it has some shortcomings for modelling social context that is considered to be a major disadvantage. Firstly, it is limited to specific relationships and context types. Secondly, it is not able to model all uncertain contexts and does not support all reasoning. Moreover, it is also not capable of managing large value of data (B. Schilit, Adams & Want, 1994; Indulska, Robinson, Rakotonirainy & Henricksen, 2003).

2.5.2 Object-Role Model

Context Modelling Language (CML) is a method of Object-Role model that exploits relationships to model context. It allows the modelling constructs to define the kinds of information, dependencies between different kinds of information and their classification (Halpin & Morgan, 2010). CAMEO implements CML, which is proposed by Henricksen and Indulska (2006), to model context. CAMEO defines three types of objects to represent social contexts: individual, neighbour and community in order to present social information components. These components are based on a predefined fact types to model the relationships. For instance, "an individual is a part of a community", to demonstrates the person of the local users of a home (Arnaboldi et al., 2014).

Despite the fact that CML has the ability to handle complicated relationships between users, uncertain and historical context data, it cannot deal with hierarchical

structure and interoperability. As result, issues may appear, when analysing social contexts that are modelled in different manners (Henricksen & Indulska, 2006).

2.5.3 Ontology-Based Model

The word *ontology* in philosophy points to the subject of being or existence. According to the AI literature, an ontology is a clear depiction of theory in an area of discourse. It contains knowledges of domains and descriptions of specific situations in a domain, which is represented in a form of vocabulary (Gruber, 1993), while, the meaning of ontology in computer science field is any shared information that is described as a set of entities, relations, functions, axioms and instances (X. H. Wang, Zhang, Gu & Pung, 2004).

A great number of social context models have adopted ontology-based-models (Biamino, 2011; Paul-Stueve & Wachsmuth, 2012). The primary advantage of implementing ontology-based in context-awareness is that it allows to interpret context to more understandable language, it also works separately of programming languages, operating systems or middleware (Gu, Pung & Zhang, 2005). Recently, several ontology standards have been founded but Resource Description Framework (RDF) (Klyne & Carroll, 2006) and OWL (Riboni & Bettini, 2011) are widely deployed.

The social context model in Yarta middleware is based on RDF and the advantage of it is that it can be expanded to accommodate more classes and properties that are related to the base classes (Toninelli et al., 2011). In comparison to Yarta, SCIMS middleware uses OWL 2 DL to model social links between people. SCIMS defines both domain-specific and upper ontology. SCIMS specifies several domain-specific ontologies. For instance, fine-grained relationship model defines the ontology for family, education-based friend, work and common-interest friend. Moreover, the proposed ontology is not limited to what has been defined but it can also be expanded depending

on the application requirement, particularly in domains such as home, office, shopping and travel. The upper ontology model specifies four first-class entities: individual, social-role, relationship and current-status (Kabir et al., 2012). Similarly, SAMOA middleware uses OWL via Jena semantic web framework to present and store user's data, such as place's profiles, and user's profiles. (Bottazzi et al., 2007). The following presents an example of a study that proposed an ontology for Facebook.

2.5.3.1 Facebook Ontology

Facebook ontology-based model, called Interaction Event Ontology (IntEO) was proposed by (Kabir et al., 2014), as shown in Figure 2.4. It can be seen that the model consists of two parts: the core event ontology and the interaction event ontology.

First, the upper part of the ontology in Figure 2.4 shows the core event ontology, which presents the most basic classes. The core event ontology was based on two different models: Description of Events (LODE) (Shaw, Troncy & Hardman, 2009), and SEM models (Van Hage, Malaisé, Segers, Hollink & Schreiber, 2011), which were the most recent designed ontology. The core event ontology contains four classes: Time, Place, Thing and Event.

Second, the bottom part of the ontology in Figure 2.4 presents the IntEO, which was based on the core event ontology. IntEO was achieved by extending the core ontology with ideas relating to interaction events and combining them with Social Context Ontology (SCOnto) (Kabir et al., 2012). As shown in the lower part of Figure 2.4, nine classes were defined. *InteractionEvent* and *Place* classes are a subclass of the Event class from the core event ontology. *TemporalEntity* class in the upper ontology was adopted from Time class in the core event ontology. *Platform, InteractionType, Topic, Relationship, SocialRole* and *Actor* classes were extended from Thing in the core event ontology, in which Relationship and SocialRole were borrowed from SCOnto ontology. The *from, to, fromRole, toRole, related, about, hasType* and *hasPlatform* are

object properties or relationships that enable to link two classes (Kabir et al., 2014).

InteractionEvent class is the main class in the upper ontology and is extended to six classes: *Actor, SCOnto:SocialRole, SCOnto:Relationship, Topic, InteractionType and Platform*. *Actor* class is inherited by *SCOnto:Person*, where all individuals are captured. Every individual who is associated with an interaction, is identified by a role, which is captured in *SCOnto:SocialRole*. The *from* property links the person who performs the interaction with the *InteractionEvent*, while the *to* property links the person who participates in that interaction. *SCOnto:Relationship* class contains information about the relationship between two users. This class is connected to the main class, which is *InteractionEvent* class, through *related* object property. *Topic* class consists of information regarding the goal or reason of that interaction. *InteractionType* capture the mean of communicating either through private message, comment or post. *Platform* class identifies different platform sources; for example, Facebook, LinkedIn, Twitter, Google+, Instant Messenger and Email (Kabir et al., 2014).

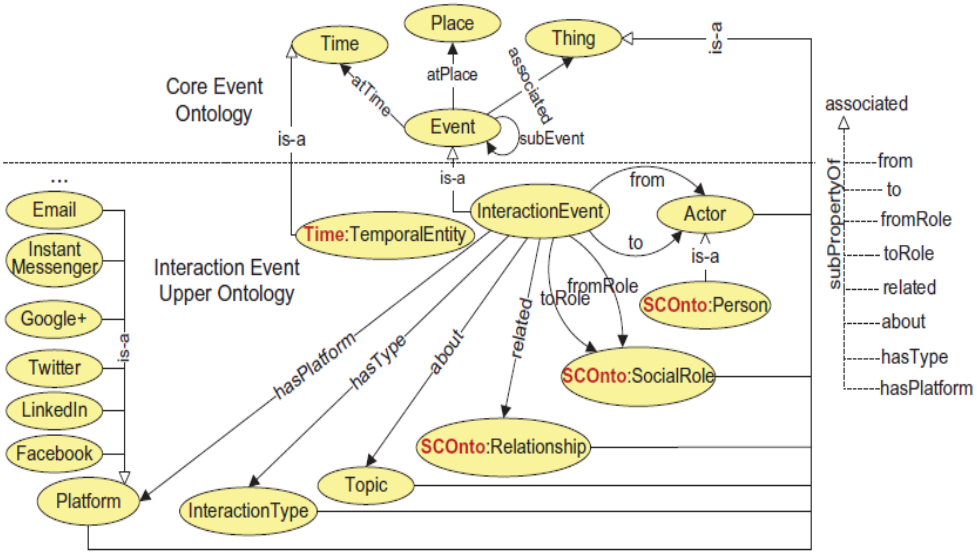


Figure 2.4: Interaction event upper ontology (Kabir et al., 2014)

2.5.4 Hybrid Model

Hybrid combines two different models allowing complex system to achieve the requirements such as, hybrid fact-based/ontological. This hybrid fact-based/ontological model is a combination of the ontology-based model and the fact-based model, via the mapping from CML modelling constructs to OWL-DL classes and relationships (Henricksen, Livingstone & Indulska, 2004). The combination of both models have the advantage of being able to model ambiguous and imperfect context information with interoperability support (Bettini et al., 2010).

2.6 Context Reasoning

At the point when a formal way is taken to deal with context model, the contexts can be handled with logical reasoning mechanisms instruments. Context reasoning can be used to check the consistency of contexts and deriving high-levels contexts that have been understood from low-levels. Context reasoning plays an important role in context-aware applications. It is obvious that high-level contexts cannot be directly retrieved from nodes; it is reasoned from sensor-driven, low-levels contexts, such as demographical information (Gu et al., 2004).

There are various types of reasoning that have been approached on uncertain context information. The following section reviews some of these approaches: Rules Reasoning, Probabilistic Logic Reasoning, and Ontology-Based Reasoning.

2.6.1 Rules Reasoning

Rules is a simple technique that has the potential of imitating the human thinking and reasoning. The model implements the IF-THEN-ELSE framework that can distinguish a high-level social context from a low-level one by listing all the rules. Rule-based

reasoning has a number of critical drawbacks. Firstly, it is not efficient for a large social network as this will increase the total time of inference. In addition, it does not support uncertain or procedural information (Choi, Park, Hyun, Lee & Sim, 2008).

2.6.2 Probabilistic Logic Reasoning

Probabilistic is a combination of reasonable statements, and probability such as, "the probability of E_1, is less than $1/3$ and the probability of E_1, is at least twice the probability of E_2, where E_1, and E_2 are arbitrary events" (Fagin, Halpern & Megiddo, 1990, p. 1). This method has been adopted in different studies: Hidden Markov (Liao, Patterson, Fox & Kautz, 2007) and Dempster-Shafer (Shafer et al., 1976). A Hidden Markov model uses observable evidence to represent the hidden state. After that, it models the transition with Markov series. Dempster-Shafer is a concept of evidence that calculates the probability of an event after combining evidences from different sensors. The ability of abstracting useful context from various sensors shows the usefulness of this model .However, a huge amount of data are required for this model to work favourably.

2.6.3 Ontology-Based Reasoning

Ontology-based reasoning is a type of semantic web language, such as OWL and RDF, which is based on a description logic. The main benefit of this model is that it automatically abstracts new knowledge depending on the defined classes and properties. The proposed middleware SCIMS (Kabir et al., 2012), Yarta (Toninelli et al., 2011) and SAMOA (Bottazzi et al., 2007) have implemented ontology-based reasoning method. In particular, the inference process in SAMOA uses description logic to distinguish between two users who purchased the same object. The defect of this reasoning mode is that it cannot deal with incomplete and uncertain contexts. Besides, it can barely handle

a dynamic social context, because when the amount of data is overloaded, performance problems arise (Liang & Cao, 2015).

2.7 Social Tie Inference

Social tie is responsible for inferring the relationships between two users. It consists of two factors: social tie indicator and inference mechanism. Social tie indicator holds information of certain relationships, such as common friends (X. Yu, Pan, Tang, Li & Han, 2011), similar interest (Zhuang, Mei, Hoi, Hua & Li, 2011) or location (Ma, Cao, Yang, Ma & He, 2014). After comprehending these relationships, inference mechanism is implemented to determine the link between the users.

The most widely accepted inference mechanisms are topology-based, ontology-based and similarity-based techniques. Topology-based technique do not require a complete topology of a social graph to extract social tie, while an ontology-based technique requires the domain ontology to infer social tie. Similarity-based technique relies on similarity measurements, which are computed to identify the relation between the users (Lü & Zhou, 2011).

Topology-based technique is used in Prometheus (Kourtellis et al., 2010), while ontology-based technique is implemented in SCIMS (Kabir et al., 2012). Similarity-based technique is adopted in MobiSoC (Gupta et al., 2009) and VENETA (Von Arb et al., 2008). Notably, MobiSoC middleware consists of an additional factor of social tie inference, which is called social state learning. In a social state learning factor, social tie strength is responsible for calculating the obtained information from social tie indicator, such as common events, similar friends, common interests or common locations (Gupta et al., 2009). SAMOA uses the traces of common locations to infer social tie. For example. if two people are in the same library for a certain amount of time, they will be regarded as having a common interest (Bottazzi et al., 2007).

2.8 Privacy

Privacy and security are the most important aspects of social context middleware. Several studies have proposed different methods for accomplishing secure middleware, such as access control, encryption and cryptography.

Access control is related to a privacy protection technique and contains a set of policies to restrict the access to information (Liang & Cao, 2015). Numerous studies have grouped access control methods into: semantic rule-based, trust-based, role and relationship-based. Semantic rule-based access control technique describes the access policies in a semantic web rule language and it represents the social knowledge based on the ontology framework (Carminati, Ferrari, Heatherly, Kantarcioglu & Thuraisingham, 2009). Trust-based access control technique collects several types of contextual information to calculate trust metrics, including relationship type, degree of separation, and trust level between users in the network (Carminati, Ferrari & Perego, 2006). Role and relationship-based access control technique takes full advantage of the role and relationship to determine who can access and distribute the data (Kayes & Iamnitchi, 2013). Access control has been implemented in several middleware, such as MobiSoC (Gupta et al., 2009), SCIMS (Kabir et al., 2012), VENETA (Von Arb et al., 2008) and Yarta (Toninelli et al., 2011). In particular, MobiSoC has a privacy statement, which retains privacy preferences. The privacy statement consists of two entities: primary and secondary. The primary entity begins the statement and the secondary entity performs the statement (Gupta et al., 2009). In comparison with the previous middleware, Yarta deters unauthorized access by performing an authentication method for access control (Toninelli et al., 2011).

Encryption and cryptography forbid outsiders from accessing sensitive information by providing secure communications. MobiSoC (Gupta et al., 2009) and VENETA (Von Arb et al., 2008) have achieved different encryption mechanisms. Especially in

MobiSoC, the performance of different mechanisms has been evaluated. The results of the evaluations indicate that RSA encryption shows a significant performance in granting location updates for authentication and integrity. AES encryption of its cipher key, however, shows the best performance when it deals with large classified messages (Gupta et al., 2009). Likewise, several developed cryptography mechanisms have been implemented in applications that involves two participants. In particular, features such as privacy-preserving interest sharing, and private scheduling can be performed with the aid of a semi-trust server (Cristofaro, Durussel & Aad, 2011).

2.9 Review of Issues

This section has given a review of the existing trials of social context-aware. The review focuses on the following aspects: social context definition, architecture, context model, reasoning method and privacy.

Table 2.1 shows the interesting outcome that all the proposed middleware have defined social context differently. It can be analysed from the mentioned middleware that social context can be classified into two main theories. First, social context is the information extracted from either physical or virtual social interactions. Second, social context describes both physical and virtual interactions. However, middleware who adopts the second theory have a complex system and service, because they are required to combine information from both cyber and physical worlds.

Table 2.1: Summary of Social Context-Aware Middleware

Middleware	Social Context Definition
CAMEO	The information that is extracted from both virtual and physical social interactions between users (Arnaboldi et al., 2014).
CircleSense	The information that describes the actual interactions between a number of users (Liang et al., 2013).

Table 2.1: Summary of Social Context-Aware Middleware

Middleware	Social Context Definition
MobiClique	The information that describes the relationships and interactions between users who are in co-locations (Pietiläinen et al., 2009).
MobiSoC	the information that identifies common links between the interactions between people and the interactions among people and locations (Gupta et al., 2009).
Prometheus	The information that describes the physical interactions between users (Kourtellis et al., 2010).
SAMOA	The information which describes the interactions between a number of users who are in a similar location.
SCIMS	A form of information that is extracted from both virtual and physical interactions between users (Kabir et al., 2012).
SocialNetwork	The information that describes both virtual and physical interactions between users (Mokhtar et al., 2009).
VENETA	The information that describes the relationships between users (Von Arb et al., 2008).
Whozthat	The data that describes the relationships and interactions between users who are in co-locations (Beach et al., 2008).
Yarta	The information which describes the relationships among users in co-locations (Toninelli et al., 2011).

Table 2.2 discusses the architecture, model and reasoning that middleware have adopted. It can be seen that all the middleware are presented in different designs. Some middleware adopted the P2P architecture, while others prefer to implement centralized architecture. It also can be seen that similarity-based and ontology-based reasoning techniques are the most deployed method in these middleware. However, the small number of middleware who have performed an ontology-based model is minimum. This could be because of the lack of a unified model.

Table 2.2: Summary of Social Context-Aware Middleware

Middleware	Architecture	Context Model	Reasoning Method	Privacy
CAMEO	P2P	Object-role	Knowledge-based	N/A
CircleSense	Centralized	N/A	Supervised machine learning	N/A
MobiClique	P2P	N/A	N/A	Enabled

Table 2.2: Summary of Social Context-Aware Middleware

Middleware	Architecture	Context Model	Reasoning Method	Privacy
MobiSoC	Centralized	N/A	Similarity-based	Enabled
Prometheus	P2P	N/A	Topology-based	Enabled
SAMOA	P2P	Ontology-based	Ontology-based	N/A
SCIMS	Centralized	Ontology-based	Ontology-based	Enabled
SocialNetwork	Semi-distributed	N/A	Similarity-based	N/A
VENETA	P2P	N/A	Similarity-based	Enabled
Whozthat	P2P	N/A	Similarity-based	N/A
Yarta	P2P	Ontology-based	Ontology-based	Enabled

Despite the fact that security and privacy are essential for applications containing sensitive information, the outcomes show that current researches of social context pay minimal attention to theses features. Social context-aware systems are interested in user's data, which is derived from both cybers and physicals. These data are instinctively sensitive; thus, an implementation of security and privacy techniques are worthwhile for data collection, inference and publication.

Chapter 3

Method

3.1 Introduction

This Chapter provides a detailed description of every procedure that has been taken in this research. It consists of two main Sections; the first section 3.2 shows an explanation of Design Science Research Methodology (DSRM), including the process model of the original method. Upon the understanding of the methodology, it has been adopted, and adjusted in order to suit the study, and answer the research questions in chapter 1. The second section 3.3 includes several sub-sections: Research Design, Proposed Architecture, Ontology Design, Conceptual Framework, Environment, and Data Requirement. First, research design describes the stages that the experimenter performed in exact detail. Second, it explains the proposed architecture of the solution. Third, it discusses the design of the ontology, and defines all the classes, and properties. Next, it explains an overview of the Framework concept. Later, it describes the laboratory environment and the components. Afterwards, it demonstrates how the data are generated, what data are collected, and how the data are presented. Finally, the conclusion gives a brief description of this chapter.

3.2 Research Methodology

This research applies the methods of Design Science Research Methodology (DSRM) proposed by (Peffers et al., 2007). The aim of the methodology is to specify the processes of DSRM, and discipline them. In addition, the study is provided with a mental model, and a template to structure the output.

The following Section 3.2.1 provides a brief description of the history of DSRM. It also shows the processes, and the iteration of the method. In addition, it presents the adoption of the method with a small modification to the name of the activities in the process model, and explains these activities in details.

3.2.1 Design Science Research Methodology

Even though DSRM has been proposed for more than a decade, it is still continuing to evolve. Today, there is plenty of information that can be found about DSRM. The function of DSRM was described by Backhaus and Murungi (2004) as "to solve problems by introducing into the environment new artifacts" (p. 103). Such a solution may include constructs, models, methods, and implementations (Nunamaker Jr, Chen & Purdin, 1990).

DSRM consists of six processes: Identify Problem and Motivate, Define Objective of Solution, Design and Development, Demonstration, and Communication. These processes are in a sequence as shown in Figure 3.1. According to Peffers et al. (2007), these steps were agreed by seven different studies. However, this methodology was adopted and refined in its processes in order to suit this research. According to Geerts (2011) most of the studies that have applied DSRM did not cover all of the activities.

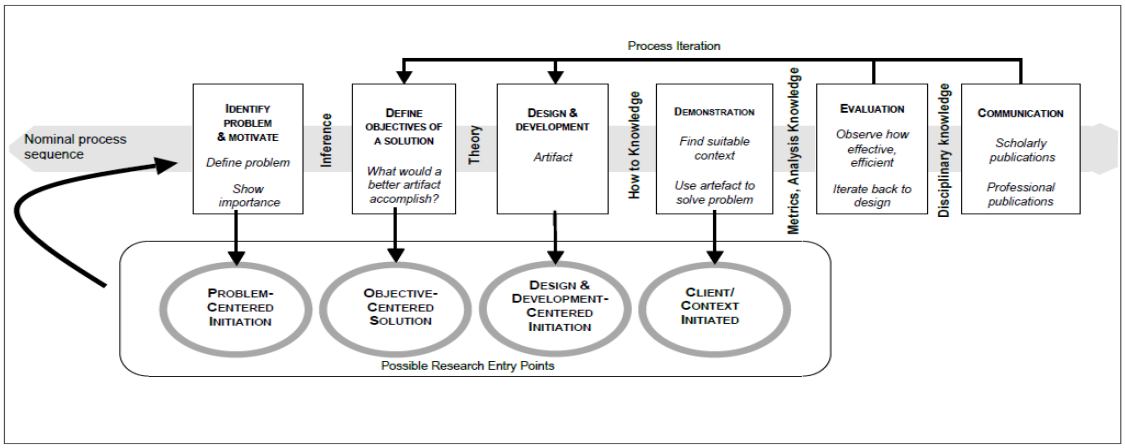


Figure 3.1: Design Science Research Methodology Process Model *Source, Peffers et al., 2007, p. 48*

DSRM is the most convenient methodology for this research, because it simplifies the iterative steps of developing gadgets, and evaluating them. Thus, the theoretical framework can be validated. The processes of DSRM as implemented in this study contains processes: Review of Issues, Design and Development, Case Study, In-Lab Experiment, and Conclusion as shown in Figure 3.2. These processes were used in this research in order to achieve the goal of the research.

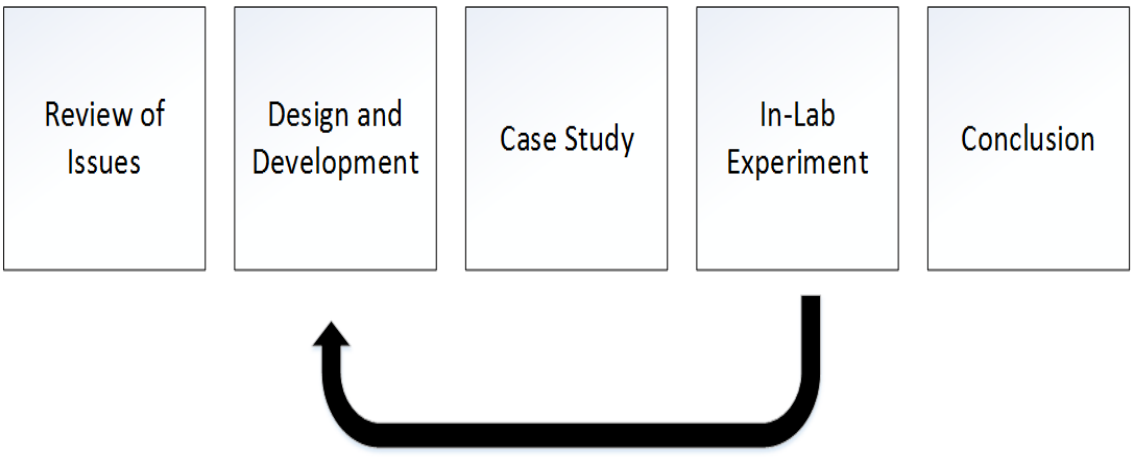


Figure 3.2: Research Phases

Review of Issues: The first process identifies the specific research problems that are extracted from multiple sources. It also represents the same meaning as Problem Identification, which is in the original model of DSRM. Peffers et al. (2007) suggested that, since developing an artifact will provide an effective solution to the problem, it might be helpful to consider atomizing the issue reasonably, so that the solution can capture its complexity. This process also shows a brief justification for the value of the solution. This helps to motivate the researcher, and the audience to agree on the solution, and comprehend the problem.

Design and Development: The second process determines the desired functionality in the artifact that includes architecture. Theoretically, designing a research artifact should probably involve constructs, models, methods, or instantiations that should be applied in any designed object to show research contribution(Peffers et al., 2007).

Case Study: The third process describes how the artifact works to solve one or two cases of the problems by showing different scenario cases. This is a part of the actual methodology of DSRM, and it is represented as Demonstration. The resources needed for the demonstration involves familiarity of the use of the artifact to solve the problem.

In-Lab Experiment: The fourth process observes, and measures the performance of the artifact, and how good the artifact is in resolving the problem. This process requires understanding related metrics, and analysis techniques. There are different kinds of evaluation methods; for instance, simulations, performance measures, surveys, or even feedbacks. However, selecting the right method depends on the nature of the artifact.

Conclusion: This is the final process in the research cycle. It is represented in DSRM as Communication. It concludes the identified problem, and its importance, the

artifact, its utilities, its design, and the effectiveness of the research to the individual. This phase is very important because it shows the knowledge contribution of the research effort.

3.3 Design and Development

This Section determines the desired functions in the framework approach. First, it details how the research was designed in order to achieve its goal. Second, it demonstrates the overview of the architecture proposed, and the tasks of each components of the architecture. Third, it discusses the design of the ontology, and the format used. Next, it summarises the concept of the framework, and the aspect of the framework in order to work as planned. Later, it illustrates the computer technologies including the machines, the programming languages, and other tools that were used to build the proposed architecture. Finally, it defines the data requirement that the research needs to be gathered, including how the data is generated, what type of data is collected, and how the data are presented.

3.3.1 Research Design

The aim of this research project is to investigate techniques for acquiring social context from OSNs. Thus, this Section shows the stages that has been designed for the research.

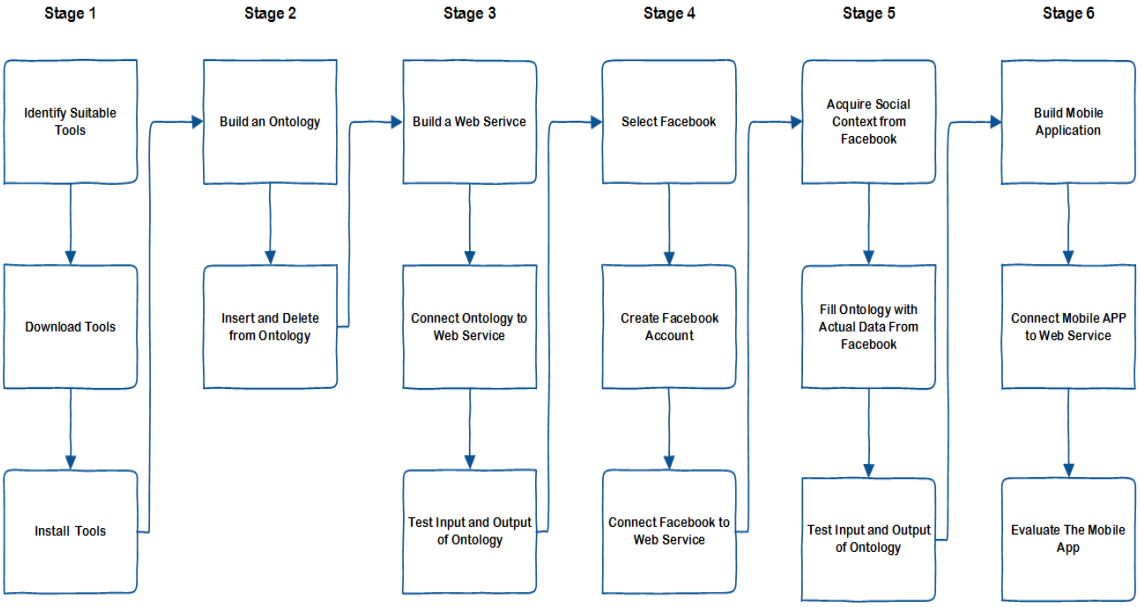


Figure 3.3: Research Stages

Figure3.3 shows six stages that the research has identified. The first stage is preliminary searching, and investigating different tools and software in order to find suitable tools for the research. This stage also determines the Operating System (OS) of the PC, and the Mobile Device. After selecting the appropriate tools, they are downloaded, and installed on the PC to prepare for the research in order to determine their usability. There are several tools that can be used in this research. However, the tools selected were chosen due to the comprehensive resources available online, and the time limitation of the research.

The second stage is to create an ontology from the beginning, using the selected tool. This stage also includes testing the proposed ontology by inserting data manually. Then, they are validated by HermiT Reasoner to determines whether or not the ontology is consistent, and identifies Subsumption relationships between classes, which is a representation of *is-a* relation. Finally, the data are deleted in order to be prepared to move on to the next stage.

The third stage is to build a web service that allows remote users to be connected

with the framework, using HTTP request: GET, and POST. The web service is initiated using the preferred programming language for the researcher. After setting up the web service, the ontology is linked with the web service to be able to use it and control it. Furthermore, random data are inserted into the ontology using the programming language. Finally, the data are presented in a basic manner to determine if the data are extracted properly.

The fourth stage is to create a Facebook account. There are two types of accounts created: a user account, and a developer account. The user account normally includes the usual user interface of Facebook, such as user profile, album, news feed, and friend list, while the developer account offers access to Facebook API allowing them to obtain social contexts. After registering the developer account, Facebook application has to be created, informing Facebook about the type of platform used by the developer in order to receive an application ID. Thus, Facebook is aware of the social contexts that are accessed by each developer.

The fifth stage is to determine the required social context data to acquire them from Facebook. As the web service is already set up, and linked with the ontology, the web service is prepared to be connected to Facebook to obtain the actual social context data. The data are presented in a simple HTML web page for testing purposes.

The sixth stage is to build a mobile application and connect it to the developed web service. The mobile application is prepared by installing Facebook SDK to enable the application to obtain the user's token from Facebook API, which are later sent to the framework. Next, the application's connection is tested using the actual data from the ontology to check whether the data can reach the web service. In addition, use cases are designed to demonstrate the applicability of the approach of the research, and to explain how, when, and why the proposed application functions may be used. Finally, the mobile application is evaluated to determine its impact on the mobile's battery.

3.3.2 Proposed Architecture

SCWS architecture is proposed in this paper to facilitate the different components of technical architecture. This architectural model of SCWS is based on the social hourglass model proposed by (Iamnitchi, Blackburn & Kourtellis, 2012), but it has been further adjusted to reflect the enabling technique. It can be seen in Figure 3.4 that the architecture consists of 5 layers: social acquisition layer, social context layer, social context modelling layer, social provision layer, and application layer.

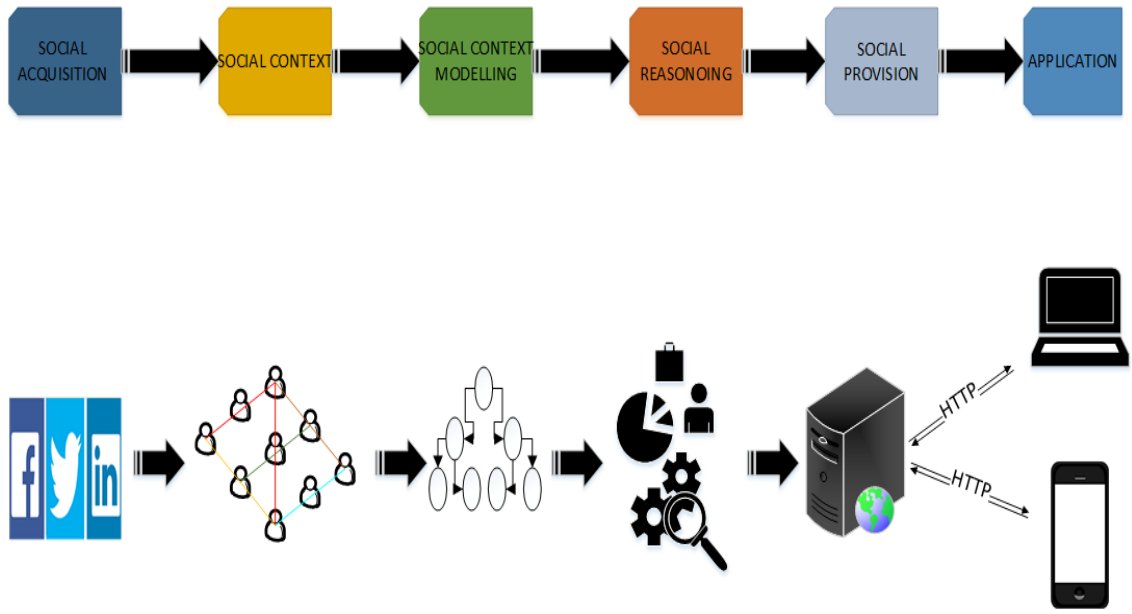


Figure 3.4: Upper Social Context Web Service Infrastructure

Social Acquisition (SA) Layer is responsible for retrieving raw context information from different sources. In the Figure 3.4 three OSNs are identified: Facebook, Twitter, and LinkedIn. These sources provide an API, which enables the developers to acquire the data by sending a set of HTTP requests.

Social Context (SC) Layer is designed to distinguish different type of social signals. According to Kourtellis (2012), social signal refers to unprepared social information such as social interaction between individuals, comments, locations. etc. These signals

are sent, and received from different sources, which have no dependence on one another.

Social Context Modelling (SCM) Layer is a way of structuring the data. When social context are obtained, they are received in multiple formats. If the data need to be changed, they have to be converted into a unified format, allowing the machine to read them. This layer acts like a parser, and a translator for the obtained data from different sources. There are various methods that have been approached such as Key-value modelling, object-role, graphical approach, markup scheme-based modelling, and ontology-based modelling.

Social Situation Recognition (SSR) Layer is performed in some of the social-aware applications. However, it can be helpful, if the acquired context is not well structured. The function of this layer is designed to merge, and analyse the social context received from the sensor.

Social Provision (SP) Layer allows outsider applications to connect to the framework through an API in order to use its resources. This service must be changeable when required depending on the environment. As Van Kranenburg et al. (2006) said, there are two types of methods that are widely used in provision. The first method is notification. This means that if the applications need a certain context data, they can subscribe to the framework, and be informed, when certain context data are updated. The second method is called polling. It indicates that the applications can request context data by queries at any time.

Application Layer shows current applications which use the service by requesting a set of social information inferences such as Mutual Friends. An example of these applications might be a website page, a desktop application or a mobile application.

Apparently, social-aware applications might not include all of the mentioned aspects, and features because a few of them are ambiguous in their representation. Typically, the method of analysing social-aware applications starts with the acquisition of different types of social contexts, followed by the modelling, and ending at the provision of

useful context to the corresponding applications. To some extent, the historical social data are needed to be stored for further use, queries, or even being visualized by the user.

3.3.3 Ontology Design

This phase shows an overview of the ontology, and discusses its content. The ontology was built based on the social context that can be acquired from Twitter, Facebook, and LinkedIn. Table 3.1 shows a description of the format used to create the ontology. Basically, the idea of the ontology is based on the expression of triples. Triples are presented as follows: *subject - predicate - object*. *Subject* expresses the resources. *Predicate* expresses the relationship between the subject and object. Table 3.2 describes the content of the ontology including all the classes, and the properties in explicit details. Figure 3.5 shows a graph of the relationship between the classes, and properties in the proposed ontology.

Table 3.1: Ontology Format

Model	Graph
Format	RDF/XML
Data	RDF
Meta data	Resource Description Framework Schema (RDFS)/ OWL, Friend-of-a-Friend (FOAF)
Identifier	Uniform Resource Identifier (URI) `http://www.semanticweb.org/xgc4811/ontologies/2016/9/socialContext`
Query Syntax	SPARQL Protocol and RDF Query Language (SPARQL)
Semantic	RDFS and OWL

Table 3.2: Ontology Design

Class foaf:Person
Continued over page

Table 3.2: Ontology Design... *(continued)*

Properties	bluetoothID, age, birthday, familyName firstName, gender, givenName, lastName, name, status, surname, imageURL
Used with	knows, isConnectedTo, isFollower, isFollowing, isFriendOf, mbox
Subclass Of	Things
Description	Person Class is the representation of people whether they are dead or alive. Example: there is a foaf:Person with a foaf:name property of "John"; this person has a foaf:image relationship to a thing reference by URI */images/me.jpg*, and a bluetoothID property relationship to a mobile device *A1:B2:A3:A5*
Class Social Context	
Subclass Of	Things
Has Subclass	Relationship
Description	Social Context Class holds any type of data that are obtained from OSNs
Class Relationship	
Subclass Of	Social Context
Has Subclass	People
Description	Relationship Class holds information of all the relationships of OSNs
Class People	
Subclass Of	Relationship
Has Subclass	Connection, Following-Follower, FriendList
Description	People Class contains the information of the relationships for only humans
Class Connection	
Properties	name
Subclass Of	People
Description	Connection Class holds information related to the connections in LinkedIn
Properties foaf:knows	
Domain	Person
Range	Person
Has Sub-Properties	isConnectedTo, isFollower, isFollowing, isFriendOf

Continued over page

Table 3.2: Ontology Design... *(continued)*

Description	knows property relates a person to another person.
Inverse Functional Property	
Properties isConnectedTo	
Domain	Person
Range	Connection
Sub-Properties Of	knows
Description	isConnectedTo property relates a person to another person who is connected in LinkedIn
Properties isFollower	
Domain	Person
Range	Following-Follower
Sub-Properties Of	knows
Description	isFollower property relates a person to another person who is a follower in Twitter
Properties isFollowing	
Domain	Person
Range	Following-Follower
Sub-Properties Of	knows
Description	isFollowing property relates a person to another person who is following in Twitter
Properties isFriendOf	
Domain	Person
Range	FriendList
Sub-Properties Of	knows
Description	isFriendOf property relates a person to another person who is a friend in Facebook
Property foaf:mbox	
Domain	Person
Range	Things
Description	mbox property relates a person to its mailbox or email
Property bluetoothID	
Domain	Person

Continued over page

Table 3.2: Ontology Design... *(continued)*

Range	Literal
Description	bluetoothID property relates a person to Bluetooth mobile device by Media Access Control Address (MAC address)
Property facebookID	
Domain	Things
Range	Literal
Description	faceboookID property relates a person to Facebook account using user ID
Property foaf:age	
Domain	Person
Range	Literal
Description	age property relates a person to its age in year
Property foaf:birthday	
Domain	Person
Range	Literal
Description	birthday property relates a person to its birthday and it is represented in dd/mm/yyyy
Properties foaf:familyName, foaf:lastName, foaf:surname	
Domain	Person
Range	Literal
Description	These properties are some person's family name
Properties foaf:firstName, foaf:givenName	
Domain	Person
Range	Literal
Description	The first name, given name properties relates a person to the name that people use to call them
Property foaf:gender	
Domain	Person
Range	Literal
Description	gender property relates a person to a string representing its gender. This value is either male or female
Property foaf:name	
Domain	Person, FirendList, Following-Follower, Connection
Range	Literal
Description	name property is a simple text that addresses something or someone
Property foaf:status	
Domain	Person
Range	Literal
Description	status property relates a person to a written statement about its feelings for the public

Continued over page

Table 3.2: Ontology Design... *(continued)*

Property hometown	
Domain	Things
Range	Literal
Description	hometown property relates a person to its hometown
Property imageURL	
Domain	Person, People
Range	Literal
Description	imageURL property relates a person to a personal image that has been made available online in Uniform Resource Locator (URL) format
Property linkedID	
Domain	Things
Range	Literal
Description	linkedID property relates a person to LinkedIn account using user ID
Property twitterID	
Domain	Things
Range	Literal
Description	twitterID property relates a person to Twitter account using user ID

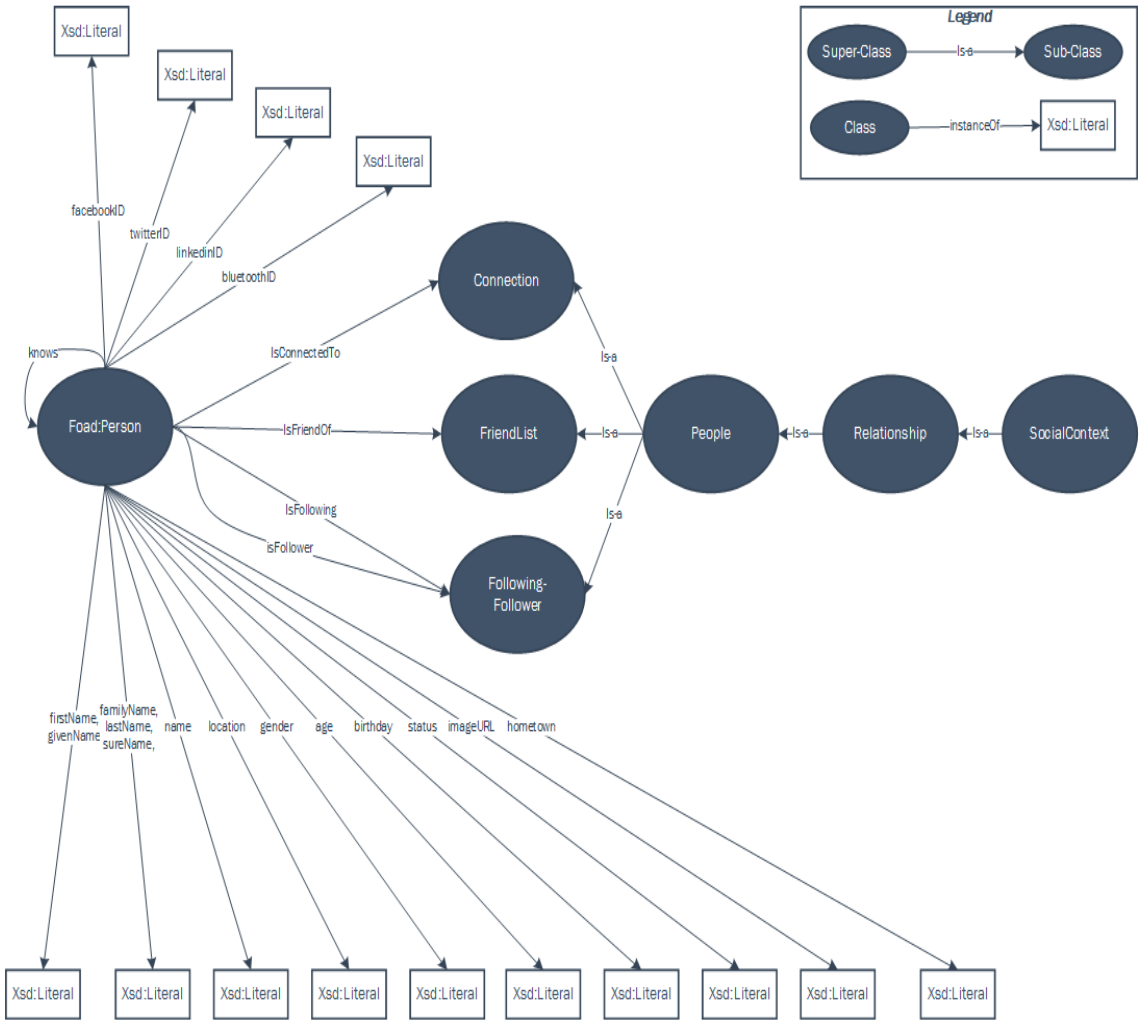


Figure 3.5: Ontology Graph

3.3.4 Conceptual Framework

The conceptual framework shows an overview of the research approach. It demonstrates the steps taken to build a framework based on the architecture proposed in Section 3.3.2. The aim of this section is to explain how the mobile application works from the beginning when the users first install it.

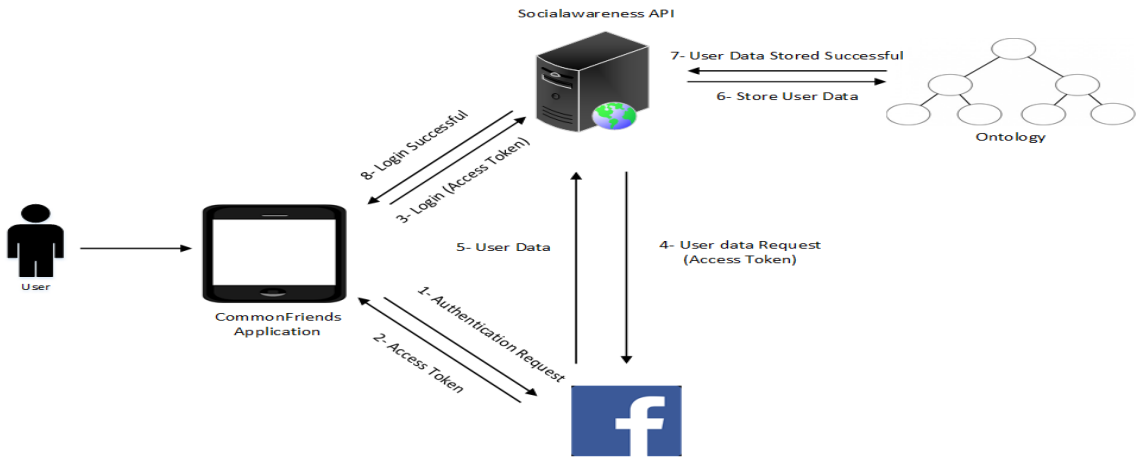


Figure 3.6: Sign-up and Login Procedure

When the users first install the application, the users have to login via the mobile application using Facebook as shown in Figure 3.6. It can be seen that the figure consists of five elements: a user, a mobile application, a Facebook's API, a server, and an ontology. These elements are combined to perform the following steps:

1. The user enters his Facebook's credentials into the mobile application.
2. The mobile application sends the user's credentials, and the application's token number that is provided by Facebook to Facebook's API.
3. Facebook's API returns a response confirming whether the credentials are valid or not. If they are valid, the API sends back the user's token number to the mobile application.
4. The mobile application forwards the user's token number to the server through API.
5. The server sends a request including the user's token number to Facebook's API in order to acquire the user's social context.
6. Facebook's API checks whether the user's token number is valid or not. If it is valid, the API sends back the social context of the user to the server.
7. The server stores the social context into the ontology, and it sends a confirmation

message with a generated token number to the mobile application.

8. The mobile application displays the message to the user, and the received token number enables the users to view the main page of the application.

These steps are taken for every user, when they sign up or login to the mobile application.

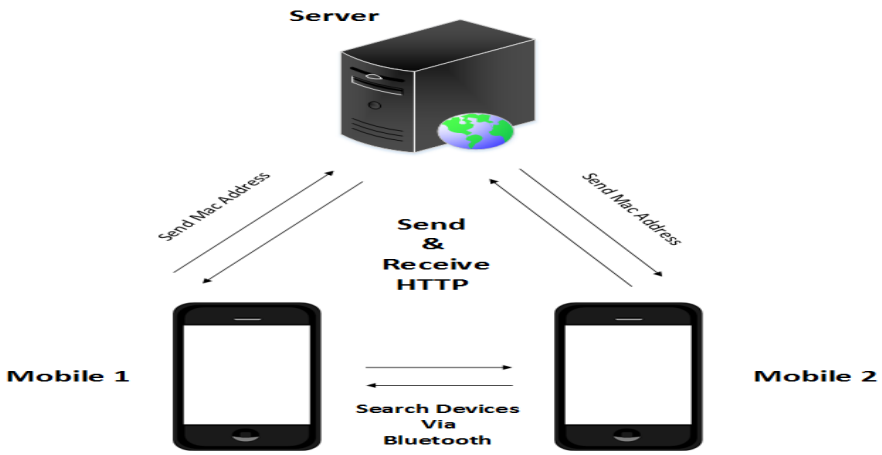


Figure 3.7: Finding Mutual Friend Procedures

After completing the above steps, the mobile application uses Bluetooth technology to search the surrounded environment looking for nearby devices as shown in Figure 3.7. It can be observed that the figure contains three elements: Mobile 1, Mobile 2, and a server; these elements functions as follows:

1. Mobile 1 searches the surrounding environment looking for Mobile 2.
2. Mobile 1 finds Mobile 2, and it sends the Bluetooth Media Access Control Address (MAC address) of Mobile 2 to the server.
3. The server compares the received MAC address with the stored MAC addresses in the ontology.
4. The server sends back 404 HTTP request NOT FOUND, If the Bluetooth's MAC address is not found, but if the server finds the MAC address, it sends back 200 HTTP request OK.

5. Mobile 1 receives a confirmation response from the server. It starts calculating the distance between Mobile 1, and Mobile 2 using RSSI.
6. Mobile 1 displays a message showing the distance of Mobile 2, and additional information; for instance, whether Mobile 1 has mutual friends with Mobile 2, or Mobile 1 is already friends with Mobile 2, or it may even suggest Mobile 1 to Mobile 2 as a new friend.

These actions are performed frequently by each mobile device. In this research the search procedure is set to locate nearby devices every three minutes.

3.3.5 Environment

The environment determines the tools, and components used to simulate the proposed architecture discussed in Section 3.3.2. This part of the research is broken down into several sub-sections that explain in details the technologies used. First, it reviews the history of the editor used to build the ontology, and it shows the advantages of using it. Second, it demonstrates the domain offered by Facebook for the developers that allows them to acquire social context. Third, it explains the specifications of the machines, and what OSs were used, and why the specified environments are selected. Fourth, it defines the software installed and used in the study. Fifth, it describes the host from third party that enables to make the framework available online. Sixth, it presents the specification, and environment for two mobile devices. Finally, it summarises the environment for building a mobile application based on OS, and the packages used to help connecting to Facebook API, and the framework. It also explains the telecommunication components used to locate other mobile devices.

3.3.5.1 Ontology Editor

There are a number of different tools, and environments for creating, and managing ontologies. One of these tools is Protégé, which provides support for the process of creating ontologies, and for the subsequent ontology usages.

Protégé was developed by Stanford Medical Informatics, at Stanford University. It is a free software, and it does not rely on other extensions or software. It provides support for the following ontology formats: RDF/XML, OWL/XML, and N3. Protégé is the most stable, and powerful tool for developing ontologies with straightforward user interface (Gennari et al., 2003).

Protégé environment has been used for building an OSN ontology in this research. The reasons for preferring to use this environment are:

- Protégé is an open source, and can be installed on different Operating System, such as, Windows, Mac, and Linux.
- Protégé does not require the internet connection. Therefore, it is accessible once it is installed on the machine.
- Protégé is uncomplicated to use, and provides multiple graphical styles for users.
- Protégé supports the import and export of different ontology formats.

3.3.5.2 Facebook API Developer

Facebook has a special domain for developers. Any person who wishes to develop an application that is connected to Facebook has to register in Facebook as a developer. After creating a developer's account, the developers are required to create a Facebook application, where they have to choose a name, and select the platform. The following Figure 3.8 shows the created Facebook application. This application allows the framework to acquire social context, and be informed of any changes the users make by connecting to Facebook's API.

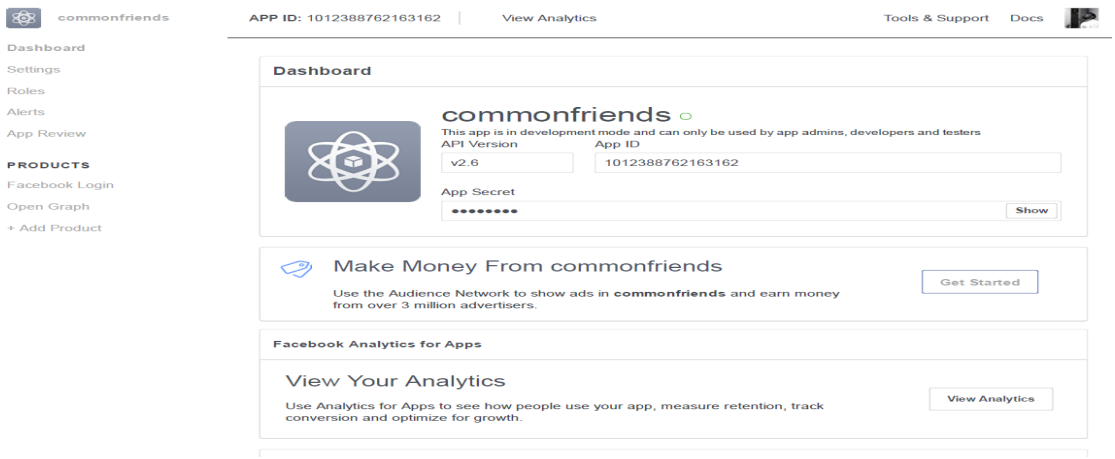


Figure 3.8: Facebook Application (CommonFriends)

3.3.5.3 Desktop

The environment of the machine used for implementing the proposed approach was on a HP Desktop that has 3.20 GHZ Intel Core i5 processor, and 16 GB of RAM. The desktop has a host, and a guest OS. The host OS is windows 7, while the guest OS is Ubuntu Linux 14.4, which was installed on a Virtual Machine (VM) using VMware application version 12.1.0. The researcher performed the model on Ubuntu OS because it is an open source. It also provides command line interface, which is more powerful, and faster than a regular user interface.

3.3.5.4 Desktop Software

The proposed model SCWS in Section 3.3.2 can be implemented using any of the programming languages available. However, SCWS was built using Python 3.5, and other frameworks; for example, Python-Django, Django-Rest-Framework, Django-Allauth, RDFlib, Facepy, SQLite. The Table 3.3 shows information about the frameworks that have been used.

Table 3.3: Desktop Software

Framework	Version	Description
Python	3.5	It is an interpreted language that has been designed based on the philosophy of code readability. There are many reasons that Python was chosen. It was ranked by *IEEE Spectrum* the fourth most popular programming languages (Cass, 2015). It is very powerful and easy to use, which makes it an advantage to learn for beginners, and experts. Its readability makes it great for programmers to not consume time trying to understand the syntax that other languages may need.
Python-Django	1.9.6	It is a web framework that is mainly used to help developers build a rapid database-driven application. It was used for many reasons; for instance, its popularity led to receive the attention of many big companies, such as Google, Pownce. etc (Forcier, Bissex & Chun, 2008). In addition, it is an open source so that it can be configured to suit any type of project.
Django-Rest-Framework	3.3.3	It is an extension of Python-Django. This framework is robust, and powerful; it has a very clear online documentation with some examples, which is an advantage for the researcher to focus on other important part of the research.
Django-Allauth	0.25.2	It is a security integration, which allows authentication, registration, and management of the user's account. It also enables users to sign-up, and login using the following OSNs: Facebook, Linkedin, Twitter, Instagram, Google. etc. This package provides a simple extra setting to connect to OSNs with no cost.
Facepy	1.0.8	It is a package that enables the application to consume Facebook's API. After getting the access token, it can act upon behalf of the users to call their social information from Facebook's Graph. It is very compelling that it can post and retrieve information context. It can be configured easily by writing a few line of code.

Continued over page

Table 3.3: Desktop Software... *(continued)*

Framework	Version	Description
RDFlib	4.2.1	It is a python library to work with RDF, and triples, which is a data structure for representing data in *subject- predicate-object* (Lu et al., 2007). It is vital for the study because it helps storing, retrieving, and managing the ontology. It provides two options for the developers to choose that they can use either pure Python code, or SPARQL, which is RDF query language that is used to search, call instances, and properties from the ontology (Lu et al., 2007). SPARQL is an important tool that saves time by allowing developers to acquire the data they need.
SQLite	3	It is a relational database that is contained in C programming. One of the advantages of using SQLite is that it is embedded SQL database engine that come ready to use with Python-Django, and it uses most of the SQL syntax. In addition, it is one of the most used databases in the world. Moreover, it can read, and write directly to the disk without additional applications. Besides, it is an open source database, which does not require extra charge (SQLite, 2016).

3.3.5.5 Hosting

PythonAnywhere is a web hosting, and an online Integrated Development Environment (IDE) that specialises in Python programming language. It was used to implement the developed web service remotely for many reasons. First, outsider applications can connect to the framework through API. The host also provides a command line interface that is similar to Linux, which was already used during the development of the web service. Another advantage of selecting PythonAnywhere is that it allows the developed application to access files. This is an important feature because the application can connect to the ontology to write and read. Furthermore, files can be transferred to and from the server through the browsers, which eliminates the need for an extra application. Finally, it provides a free service for only one application, which is adequate for the study.

3.3.5.6 Mobile Device

Two mobile devices were used in the research to complete the final stage in the proposed architecture. The specifications of the mobile devices are shown in Table 3.4. The first, device was a Nexus 5 by LG, which has 2.26 GHz quad-core Qualcomm Snapdragon 800 processors with 2 GB of RAM. Nexus 5 comes with 2300 mAh non-removable battery. The OS of the mobile device is Android 6.0 Marshmallow. The connectivity includes Bluetooth version 4.00. The second device was a Nexus 6 by Motorola, which is powered with 2.7 GHz quad-core Qualcomm Snapdragon 805 with 3 GB of Ram. Nexus 6 comes with non-removable battery 3220 mAh. The mobile device runs on Android 6.0 Marshmallow. It also comes with Bluetooth version 4.10.

Table 3.4: Specification of Mobile Devices

Device	OS	CPU	Chipset	RAM	Battery	Bluetooth
Nexus 5	Android 6.0 Marshmallow	2.26 GHz quad-core	Snapdragon 800 processors	2	2300 mAh Non-removable	v 4.00
Nexus 6	Android 6.0 Marshmallow	2.7 GHz quad-core	Qualcomm Snapdragon 805	3	3220 mAh Non-removable	v 4.10

3.3.5.7 Mobile Software

Google provides an IDE in order to develop a mobile application in Android environment. The mobile application was built using Android Studio 2.2.2.0 with SDK version 23. The IDE is an open source code editing, and debugging tool. The advantage of using it is that it includes Layout Editor, APK Analyzer, Instant Run, and Optimization for all Android devices to eliminate tiresome tasks. In addition, two packages were used in the development of the mobile application: Facebook SDK, and Volley. Table 3.5 shows more details about the packages.

Table 3.5: Mobile Software Packages

Dependencies	Version	Description
Facebook SDK	4	It allows users to sign in an application with Facebook login. When the users login into the application, the application can grant permission to access information on the users' behalf. It is very necessary to implement Facebook SDK because it can take advantage of reading, and writing to Facebook API. The SDK also enables users to login to Facebook, even if the users do not have the Facebook application installed on their devices.
Volley	1.0.0	It is an HTTP package that makes the network management in Android easier, and faster. It offers many benefits that it allows to schedule network request automatically. It supports prioritising the requests, which is an important feature. It has also the ability to cancel a single request, or a group of requests. Another vital feature is that it is customizable to suit any type of projects.

3.3.5.8 Mobile Bluetooth (RSSI)

Received Signal Strength Indication (RSSI) may become one of the most used technologies to estimate the distance between machines because it is low cost (Bachrach & Taylor, 2005). It is used in this research to measure the distance between mobile devices due to the ability of Bluetooth component to estimating the signal strength. The signal strength can be explained as: the closer the distance of the signal to the receiver device, the stronger the RSSI at the received device, and the opposite is also true (Jung, Kang & Bae, 2013). Table 3.6 shows a model adopted for RSSI in order to measure the distance.

Table 3.6: RSSI Range *Source, Carrera Carbó, 2012, p. 21*

Range [dBm]	Distance [Meters]
RSSI > -45	0
-45 > RSSI >= -51	1
-47 > RSSI >= -51	2
Continued over page	

Table 3.6: RSSI Range... *(continued)*

Range [dBm]	Distance [Meters]
-51 > RSSI >= -54	3
-54 > RSSI >= -58	4
-58 > RSSI >= -61	5
-61 > RSSI >= -68	6
-68 > RSSI >= -71	7
-71 > RSSI >= -76	8
-76 > RSSI >= -80	9
-80 > RSSI	10

3.3.6 Data Requirement

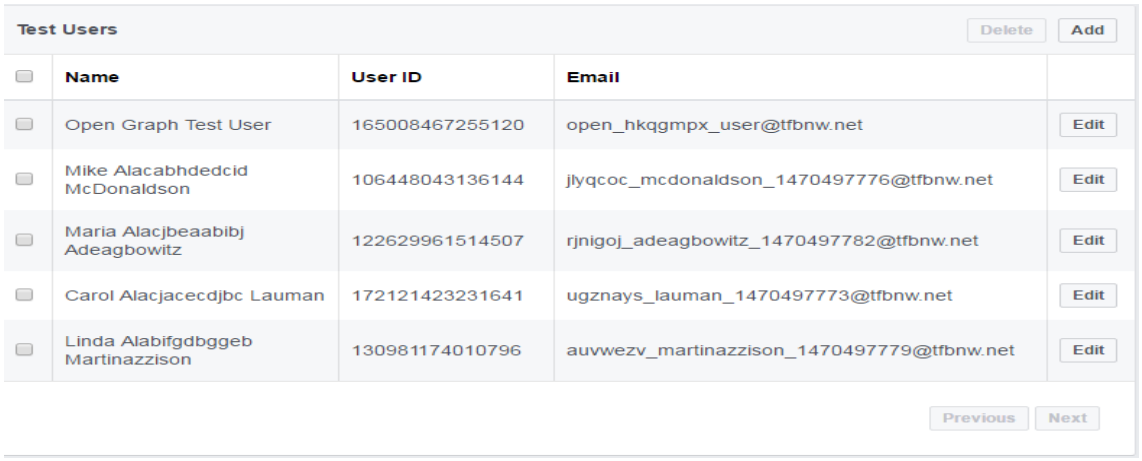
This Section explains how data is processed by the system from input to the final output. There are several type of data that are required for the research. Due to the huge volume of data that can be obtained from Facebook. The primary focus of this Section is to clarify what exactly has been accomplished. It discusses how the data are generated, what types of data are collected, and how the data are presented.

3.3.6.1 Data Generation

Data generation is an essential step that helps and supports the research experimental system design. The goal of data generation is to demonstrate the way of generating the data.

The method of generating data requires several experimental accounts in Facebook. The researcher created four tester accounts in Facebook in order to simulate real life scenarios. Accordingly, Facebook provides a virtual test environment that looks the same as the real environment. It enables developers to test their application using a so called *Test Users*. A test user is a special Facebook account, hidden to real accounts, which can be created within an application. The main objective of this account is to perform a manual or automated testing of that application's Facebook integration

(Facebook, 2016). Figure 3.9 presents the test users participated. The developers can create at maximum 2000 Test users per Facebook application, in which they can simulate real activities, such as posting images, text, videos, likes posts, comments. Then, these activities can be acquired from Facebook's API. Therefore, the researcher created random information, and assigned some friends to several users. For example, tester account 1 is a friend of tester account 2, while tester account 2 is a friend of tester account 3. Tester account 4 is not a friend with any of the accounts. Thus, Tester account 1, and account 3 have mutual friends but they are not friends.



Test Users

Delete Add

Name	User ID	Email	
Open Graph Test User	165008467255120	open_hkqgmpx_user@tfbnw.net	Edit
Mike Alacabhdedcid McDonaldson	106448043136144	jlyqcoc_mcdonaldson_1470497776@tfbnw.net	Edit
Maria Alacjbeaabibj Adeagbowitz	122629961514507	rjnigoj_adeagbowitz_1470497782@tfbnw.net	Edit
Carol Alacjacecdjbc Lauman	172121423231641	ugznays_lauman_1470497773@tfbnw.net	Edit
Linda Alabifgdbggeb Martinazzison	130981174010796	auvwezv_martinazzison_1470497779@tfbnw.net	Edit

Previous Next

Figure 3.9: Test Users

These accounts are connected to Facebook from different mobile devices. The experimenter used two mobile devices to login to Facebook via SDK. The SDK asks the user, if they are willing to permit the application to acquire the information of the user. Once the user approves the request, the application receives a token number from Facebook that enables the application to access the information of the user. This token also comes with an expiration date, but during its validity it can be used many times as long as the user has not removed the associated application.

Moreover, the mobile devices were used to locate the other device using one of the communication components. Due to the restriction of the size of the laboratory

space, the researcher used *Bluetooth* technology as a method of searching for devices in the surrounding environment, because every vendor of the Bluetooth component in any device has a unique identifier called MAC address. Technically, the mobile device searches for MAC addresses of the active Bluetooth. If the MAC addresses are found, they are sent to the server through the API to detect whether the address belongs to anyone whose information is already stored in the ontology. If the server found one of the received MAC addresses existed in the ontology, the server recalls both the sender, and the existing social context from the ontology. This information is compared in order to find similar interests. If both users share the same friends, the server sends information to the mobile application regarding the users, and what they have in common. It also notifies the users if their friends distance is as close as 5 meters. Likewise, the users are notified of new friends who use the mobile application, if they are within their area. However, the server does not inform the user if it cannot find the MAC address in the ontology.

3.3.6.2 Data Collection

The main objective of the data collection is to determine the format and the type of data extracted for this study. All the data collected from Facebook API are received in the form of JSON as showing in Figure 3.10. These data are later categorized, and stored into the ontology using RDF format to make them easier to read for human, and machine.

```
{
   "picture": {
      "data": {
         "is_silhouette": false,
         "url": "https://fb-s-d-a.akamaihd.net/h-ak-xpa1/v/t1.0-1/p50x50/10850055_10154953717815112_1867885702953593889_n.jpg?
oh=70add24009490b7ffa359d49f4347264&oe=58FCBD81&__gda__=1492868701_9c94a4643dd58f473327753620f7a800"
      }
   },
   "id": "10157182537340112"
}
```


Figure 3.10: JSON Format

In order to acquire Facebook data, it is important to receive the user's token first. This happens once the users grant the mobile application permission to be linked with their Facebook's accounts. The token number is stored in the database. Then, the server can send a request to Facebook's API in order to obtain their social context.

There are many different types of context in Facebook. For instance, users can upload images, and videos, to post texts, and comments, and so on. The research, however, did not consider all the types of data in Facebook. It only focused on the following social context: *public profile* , *email*, and *user friends*.

User friends is not accessible for being considered as private data, despite the fact that the main purpose of the research was to obtain the user friends. Thus, mutual information can be identified. According to Facebook (2016), the API version 2.0, and above, the developer has to request `user friends` permission from each user in order to obtain their information. Another technique to acquire user friends is for the developers to fill a form on Facebook's developer website outlining the reasons why the information is required. The form is delivered to Facebook's administrators, where they decide whether to accept it or not based on the reasons provided.

Nevertheless, there is a different way of obtaining the user's friends context, Facebook offers another API way to obtain the user's friends, which is called `taggable friends`. The API provides similar information to the official user friends API but without exposing any private information, such as, email or date of birth. It only consists

of *full name*, *profile's picture*, and *taggable ID*. Hence, the ontology was developed according to the social context that can be obtained.

But not only Facebook data are required in this research. Another type of data that have been collected are *Bluetooth* MAC address. Due to the uniqueness of the MAC address, it is useful to distinguish between one device, and another. Additionally, it provides helpful information that shows the RSSI. This can be used to measure the distance between different devices.

3.3.6.3 Data Presentation

The data are only presented to the users in the mobile application. Android OS provides a very convenient method called Notification allowing the developer to notify the users within the application. The notification may consist of sounds, a vibration, and a banner view. This is very useful as the users do not have to check their mobiles frequently for recent information. When the application receives information that may interest the user, it notifies the user instantly.

3.4 Conclusion

DSRM methodology was adopted by this research. The original methodology consists of six activities, but this research has refined these activities, and renamed them to the following: Review of Issues, Design, and Development, Case Study, In-Lab Experiment, and Conclusion.

Based on the above methodology, the researcher divided the activity of Design, and Development into several sections. The first section explained the six stages that were taken by the researcher to answer the research questions.

In addition, SCWS architecture was proposed, containing five layers: social acquisition, social context, social context modelling, social provision, and application.

These layers were demonstrated in exact details that started from acquiring the data from Facebook until providing the data to outsider applications.

It also discussed Protégé editor that helped to build the ontology. This ontology consisted of seven classes that addressed the relationships of Facebook. After that, Python was used to build the framework based on SCWS architecture. Java also was used to develop an application for mobile phones. Finally, it illustrated how the mobile phones were used to generate, collect, and present the data.

In the following chapter, the results of the approach and findings will be shown in a visual and graphical manner.

Chapter 4

Findings

4.1 Introduction

This Chapter will report the findings of the developed approach and how they are presented in scenario cases as mentioned in Chapter 3. This Chapter contains two main Sections: Case Studies, and In-Lab Experiment. Section 4.2 presents three different scenario cases for different situations. This helps the reader to understand the theory of the implemented approach. It also consists of two sub-Sections that presents the finding of Data Generation & Collection, and Presentation. Section 4.3 tests the performance of the developed mobile application, and reports the finding of the impact of the mobile application on the battery. Finally, the conclusion section explains the major findings of this chapter.

4.2 Case Studies

The key purpose of a case study is to use it to demonstrate the applicability of the research approach. Three case studies are reported in different situations. In the first situation, two people who work for different firms met at least once a week, but they

did not know that they had a common friend on Facebook. The second situation is completely different, it is about two classmates who share one elective subject. One of them was at the library of the university, and realized that they need help in the assignment but they do not know if their classmates was nearby. The final situation reports on a person who enjoys doing a certain habit but they want to find someone else who also enjoys the same habit.

4.2.1 Case 1: Mutual Friends

Mike McDonaldson is a project manager in an enterprise-oriented software company. Mike is married to Maria Adeagbowitz. He uses different kinds of OSNs, but most of the time he prefers to use Facebook as a way of communicating with his friends. He also uses Facebook to post on friend's wall page, and upload pictures, and videos. Mike has two friends in Facebook: his wife Maria Adeagbowitz, and Carol Lauman.

Mike has two projects in process and he meets with his clients at least once a week. The relationship between Mike and his clients is kept strictly professional, in that they only discuss their projects. One of the clients named Linda Martinazzison has two friends on Facebook from her high school; one of them is Mike's wife Maria, and the other one is Carol.

However, Mike and Linda do not know that they share the same friends on Facebook. Mike wanted to find out who are his mutual friends on Facebook. Therefore, Mike found and downloaded a mobile application called *CommonFriends*. In the meantime, Linda also had CommonFriends application installed on her phone. So both Mike, and Linda receive a notification on their phones to let them know that they share two friends, which are Maria and Carol, as soon as they turn the Bluetooth on in their phones.

4.2.2 Case 2: Already Friends

Carol Lauman and Mike McDonaldsonis are classmates in one elective subject, friends on Facebook, and students at the same university, but they are doing different bachelor degrees. Carol usually goes to the library to do her assignments. Then, she realizes that she needs help on her elective subject's assignment from Mike, knowing he is an A star student. So she wants to know whether or not Mike is near her.

Carol knows an application called *CommonFriends* that allows her to find nearby Facebook friends. She has downloaded, and installed the application on her phone to determine the distance from Mike. In the meantime, Mike had the same application installed on his cell phone. The application automatically asks for permission to activate the Bluetooth to find nearby friends. Thus, Carol easily locates Mike through the application.

4.2.3 Case 3: New Friends Suggestion

Carol Lauman has a morning part time job as a barista in a café located at the city centre of Auckland. Carol loves reading, so after work she visits the public library daily to read books, and gain general knowledge but she does not have friends who share the same hobby with her. Therefore, she discovers a mobile application called *CommonFriends* to find out who else comes to the same library by showing the distance of people who use this application.

Meanwhile, Maria Adeagbowitz also enjoys reading books and learning general information. She also goes to the public library at the same time as Carol, and she would like to practise her hobby with someone who enjoys reading as much as she does. But, she cannot go up to someone randomly and introduce herself. So Maria used CommonFriends application that she has previously installed on her phone, and granted it permission to turn on the Bluetooth in order to find out nearby people.

Both Carol, and Maria noticed that the application on their phones notify them every day about each other because they are nearby. Thus, they add one another as friends, and became close friends that share the same hobby, and read together.

4.2.4 Data Generation and Collection

The experimental data were generated, and collected using the approach explained in chapter 3 in Section 3.3.6. The most important aspect of the data generation is that every user must have a mobile phone with the proposed application CommonFriends installed. Then, each user must have a Facebook account in order to be able to use the application. In addition, the type of data collected in this approach are the same for every user but with different context.

This Section demonstrates the steps when the users install the developed application for the first time. First, it explains the initial page of the application, followed by Facebook login for the user to login. After that, it shows Facebook's permissions that required to be accepted by the users. Then, the application requests the user permission to use Bluetooth. Besides, it describes in details the application user interface, and the functions of each button. Moreover, it shows a brief description about the data that have been stored in the ontology for only one user. Finally, it illustrates how the database has been used in the approach.

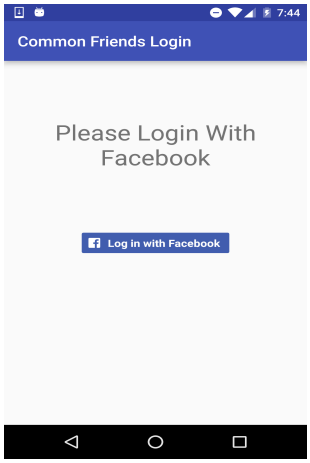


Figure 4.1: CommonFriends Initial Page

Figure 4.1 shows a screen-shot of the first page, when the users open CommonFriends application. The page contains a message requesting the users to login with their Facebook account, and a blue button in the middle of the screen. Once the users click on the button, it leads them to Facebook's login page.

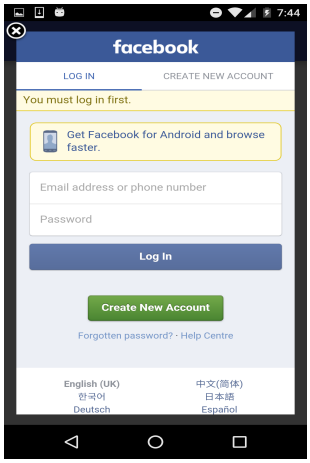


Figure 4.2: Facebook Login Page within CommonFriends Application

Figure 4.2 shows Facebook's login page, which is managed by Facebook's SDK. The login page shows two fields: email, or phone number, and password that requires the users to fill out with their credentials in order to login. If the users do not have a Facebook account, they can create a new account. To create a new Facebook's account,

the user can click on either the top right tap under Facebook's logo, or the green button at the bottom of the screen.

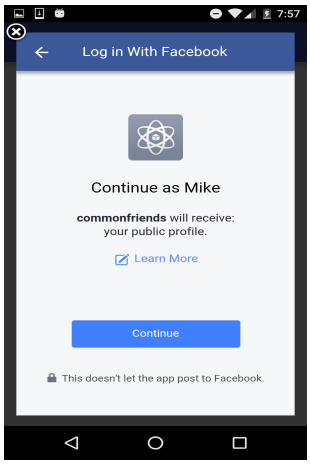


Figure 4.3: Public Profile Permission from Facebook

Once the users enter their Facebook's credentials to login, Facebook shows a warning dialogue to the users that asks them to grant CommonFriends application the permission to receive their public profiles as shown in Figure 4.3. The Figure shows three options that the users can accomplish: First, if the users wish to withdraw from the whole procedure, they can click on the X-mark at the top lift of the screen. Second, the users can learn more about what permissions are included by clicking on the URL link Learn More in the blue colour. Finally, the users can approve the procedure by clicking on the blue button at the bottom of the screen to continue logging in. After receiving the user's approval, the homepage of the application is showed.

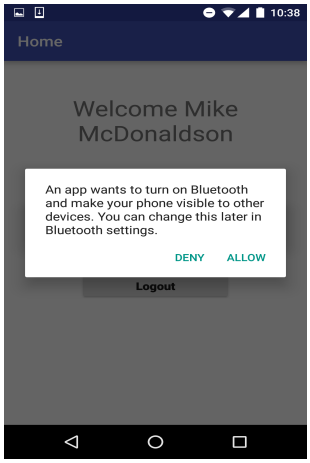


Figure 4.4: CommonFriends Application Request Bluetooth Permission

The homepage of CommonFriends application shows a pop up message immediately after a successful login to Facebook that requests the user's permission to turn Bluetooth on, and make the device visible for other devices as shown in Figure 4.4. If the users click on either allow or deny the permission, the users have the ability to amend the setting of the Bluetooth in the future.

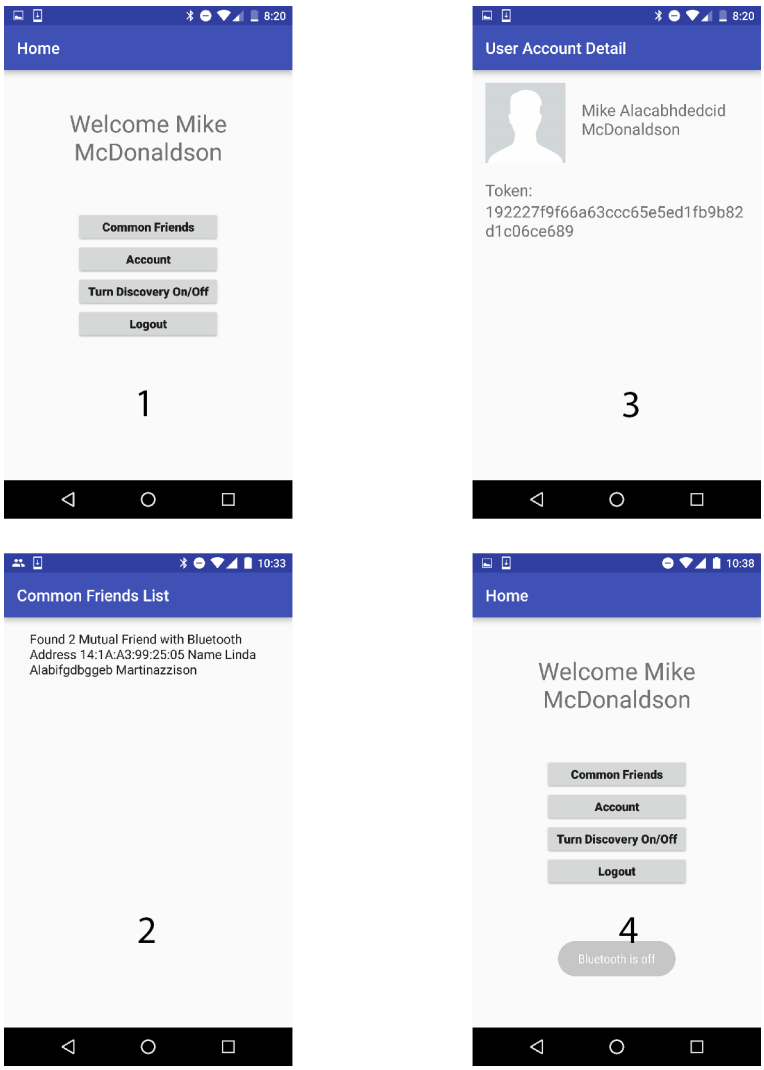


Figure 4.5: CommonFriends Application

Figure 4.5 shows four images of CommonFriends user interface. The first image is the homepage of the application. It consists of a title that welcomes the users with their full names, and four buttons: CommonFriends, Account, Turn Bluetooth On/Off, and Logout.

When the users click on the first button CommonFriends, it leads them to a page called *CommonFriend List*. This page lists all the people found nearby people who have mutual friends with the user.

The second button Account presents a Facebook profile picture of the user at the top

left of the screen, full name next to the picture, and the framework's token number of the developed approach under the label Token as shown in image 3 in the above Figure.

The third button Bluetooth On/Off turns the Bluetooth off, if it is active as shown in image 4 in the above Figure, and turns the Bluetooth on, if it is deactivated. If the Bluetooth is off, and the users turn it on, the same message shown in Figure 4.4 is presented.

The last button Logout disconnects the users from Facebook and the framework, and directs them to the page with Facebook login button as shown in Figure 4.1.

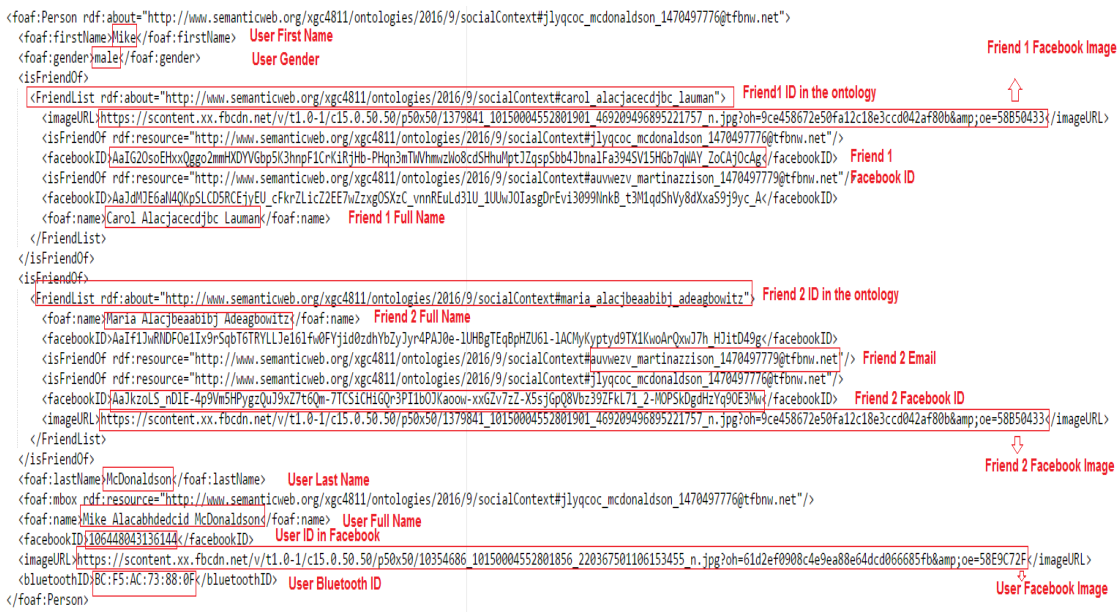


Figure 4.6: Facebook Data Stored in Ontology

Figure 4.6 shows a part of the ontology used in the study that is filled with the social context obtained from Facebook. As shown in the above figure, the ontology is represented in RDF format, and the social context of each person is stored between *foaf:Person* tags. The social context acquired from Facebook is the user's profile that includes their first name, last name, full name, gender, user's ID, user's profile picture, and email.

In addition, it can be seen that the above Figure 4.6 the user in the ontology has two Facebook friends: Maria Adeagbowitz, and Carol Lauman; they are stored between *isFriendOf* tags within *foaf:Person* tags. The contexts of the user's friends are ID, full name, email if it is public, and profile's picture.

Furthermore, in order to represents the users with logical relations within the ontology, each user between *foaf:Person* tags is identified by its email acting as a unique identifier, while user's friends are marked by their full names. For example, the space between the first, middle, last name is replaced with underscores "_", `Maria Alacjbeaabibj Adeagbowitz` is transformed to become `maria_alacjbeaabibj_adeagbowitz`.

Moreover, The MAC address of the mobile device of the user acts as an identifier between users. This MAC address is stored between *bluetoothID* tags within *foaf:Person* in the ontology. The above Figure 4.6 presents an example of the bluetooth ID of the user.

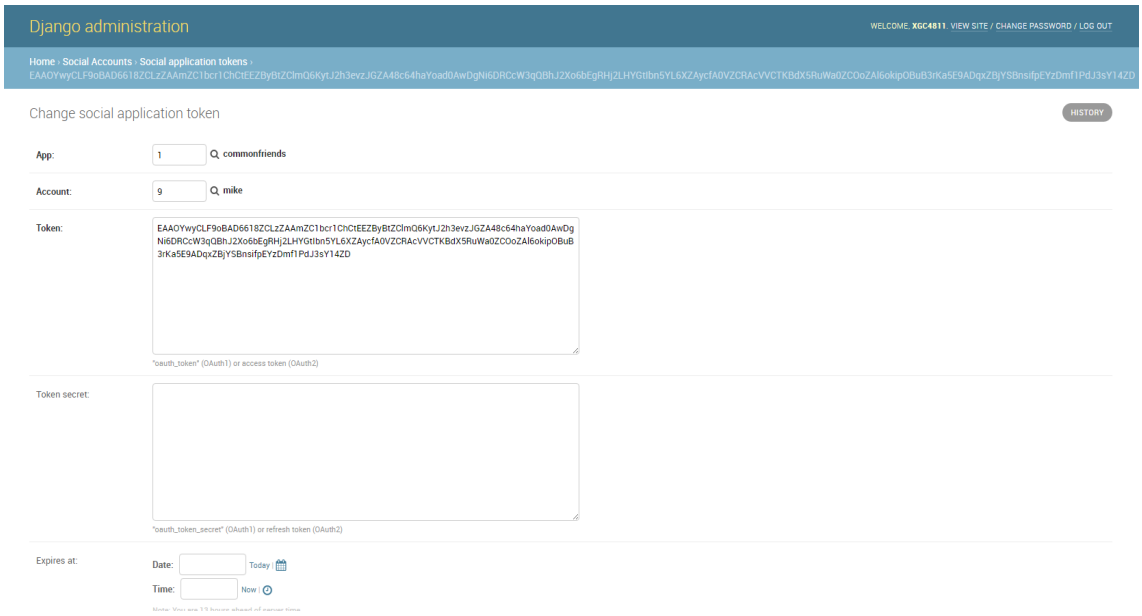


Figure 4.7: User Token Stored in Database

Figure 4.7 shows a screen-shot of the information stored in SQLite database, when

the user logs in to Facebook successfully through the mobile application. It can be seen that there are three fields: App, Account, and Token. The first field App determines the Facebook application *commonfriends*, which is created in the Facebook developer website to allow the framework to acquire the social context on the user's behalf as discussed in Section 3.3.5.2. The second field Account relates the social context to a particular user by their ID within the database users table. The Token field conserves the user's token that was received from the developed mobile application, when the user logged in to Facebook.

4.2.5 Data Presentation

The main point of this Section is to present screen-shots of the notifications received during the implementation. Each image represents a use case as explained in Section 4.2. The first use case summarises how the users can find mutual friends. The second use case describes how the users can find their Facebook friends by using Bluetooth RSSI. The last use case shows how the users can make a new friend.

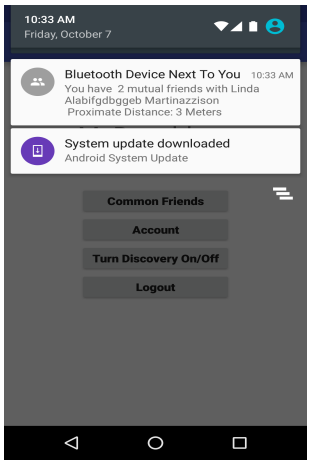


Figure 4.8: Notification of Mutual Friends Found

As can be seen in Figure 4.8, CommonFriends Application notifies the users when a mutual friend is found. This scenario is based on the Use Case 1 in Section 4.2.1. The

received notifications are displayed in the notification centre at the top of the screen. This notification informs the user that the user has two mutual friends with another user named Linda. It also notifies the user about the distance between them, which in this case is three meters.

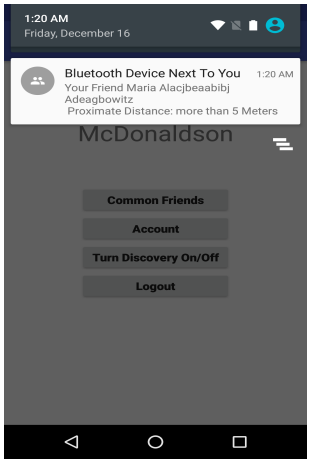


Figure 4.9: Notification of Already Friend

In addition, Figure4.9 shows a notification from CommonFriends application that warns the users that a friend is nearby. This scenario explains the situation based on Use Case 2 in Section 4.2.2. It can be seen that the notification informs the user about a friend named Maria who is 5 meters far.

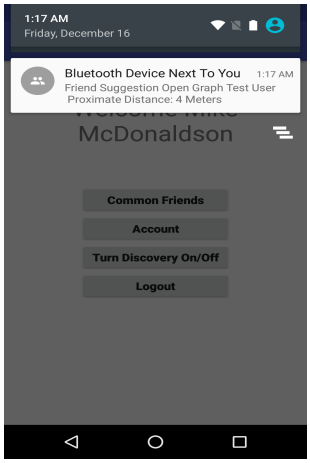


Figure 4.10: Notification of Friend Suggestion

Moreover, CommonFriends Application notifies the users, when someone who uses the application is nearby as shown in Figure 4.10. The context of this case is shown in Use Case 3 in Section 4.2.3. As shown above, the application suggests to the user a new friend. It displays the distance of the friend suggestion as well, which is 4 meters. In the next chapter 5 will discuss the above findings in detail.

4.3 In-Lab Experiments

It was very challenging to evaluate the Restful web service because it was deployed and implemented on a third party service, which is virtual over the internet. Therefore, the performance of the service could not be monitored, and the researcher skipped the evaluation of the server.

However, the mobile application was evaluated using several methods. The researcher focuses on the impact of the application on the battery, and tests the actual functions of the application. Section 4.3.1 shows the approaches of previous studies that attempted to evaluate the effect of the application on the battery.

4.3.1 Mobile Application Impact on Battery

The mobile application was evaluated in the research to validate the performance, and provide a better user experience; two evaluation methods have been adopted: Firstly, this study has used a diagnostic tool called Qualcomm's Trepn Profiler (QTP) to evaluate the performance, and the power consumption in real-time of Android applications. According to Ferreira, Kostakos and Dey (2015), QTP was used to evaluate mobile AWARE application and measure the following: the power consumption in Milliampere (mA), and the processor load(%, averaged to include all cores) of each high-performance sensor. The test was 10 minutes long and implemented at two different sampling 5, and 10 Hz. During the test, they kept the device's screen off, while 3G, Wireless, and

Bluetooth were active. However, this research has used the same tool and performed four sessions. Each session has been observed three times for different time periods: 15, 30, 60, and 180 minutes. The researcher used Nexus 6 specified in section 3.3.5.6 during the inspection, the mobile's screen, and 3G are kept off, while Wireless, and Bluetooth remained enabled; Table 4.1 shows more details about the sessions, and the results observed.

Table 4.1: Application Testing Results

Session No.	**Time Period [minutes]**	**Battery Consumption [%]**	**Battery Power [mWh]**	**CPU Load (Normalized) [%]**	**Memory [MB]**
1	15	1.00	100.88	4.34	4.84
2	15	1.00	429.04	4.62	4.84
3	15	1.00	388.8	4.47	4.84
4	30	2.00	374.65	4.22	4.84
5	30	2.00	356.39	4.30	4.84
6	30	2.00	363.1	4.39	4.84
7	60	4.00	345.61	4.43	4.84
8	60	4.00	332.24	4.11	4.84
9	60	4.00	330.48	4.24	4.84
10	180	7.00	346.5	4.56	4.84
11	180	10.00	573.97	5.47	4.84
12	180	19.00	851.02	7.04	4.84

Secondly, this research has followed a method that was implanted in a study aimed to understand the effect of battery life on the interaction of the user with the device. The study developed an application called Interactive Battery Interface (IBI) to collect some data such as, Battery usage, Network usage, Processor usage, Application usage, and Phone usage. The IBI was installed on all the participant's devices who were aged between 20 to 45. The study provided two equations for calculating the application impact on battery: $\Delta\ battery$ 4.1, and Battery Depletion Rate (BDR) 4.2. $\Delta\ battery$ equation is as follows:

$$\Delta\ battery\ (\%) = \sum_{i=1}^{n} W_{app\ i}(\%/s) \times elapsed\ times\ (s) \tag{4.1}$$

where $\Delta\ battery$ represents how much the battery was depleted by the active applications . W_{app} is the weight of the application in (%/s), when the application session started in the foreground, until it is closed or sent to the background. The $elapsed\ time$ is the time for the whole session, and it has to be equal when calculating for each application.

In addition, the study also calculated BDR 4.2 to recognize the impact of the IBI application on the battery. This equation was gauged in two scenarios: before, and after the installation of the application because it could provide useful information, such as how fast the battery runs out (Ferreira, Ferreira, Goncalves, Kostakos & Dey, 2013). Thus, the usage of the battery for the developed application could be identified, in a situation that the application causes a big impact on the battery. BDR equation is as follows:

$$BDR\ (\%/h)\ = \frac{1}{n} \sum_{i=1}^{n} \frac{battery\ change_i}{elapsed\ time_i} \tag{4.2}$$

where n is the number of battery uptime intervals, $battery\ change$ is the difference in battery percentage, and $elapsed\ time$ is the length of the uptime interval in hours. All of the above equations are, however, calculated after getting the mean of the results presented in the above Table 4.1; the averaged results are shown in Table 4.2.

Table 4.2: Average Application Testing Results

Session No.	**Time Period [minutes]**	**Battery Consumption [%]**	**Weight of Application W_{app} [%]**	**Battery Power [mWh]**	**CPU Load (Normalized) [%]**
1	15	1.00	0.07	76.56	4.48
Continued over page					

Table 4.2: Average Application Testing Results... *(continued)*

Session No.	Time Period [minutes]	Battery Consumption [%]	Weight of Application W_{app} [%]	Battery Power [mWh]	CPU Load (Normalized) [%]
2	30	2.00	0.07	182.35	4.30
3	60	4.00	0.07	336.11	4.26
4	180	12.00	0.07	1771.49	5.69
Calculation					
Total of Time Period			285 minutes		
Total of Battery Consumption			19.00 %		
Δ Battery			19.00 %		
BDR			4 %		

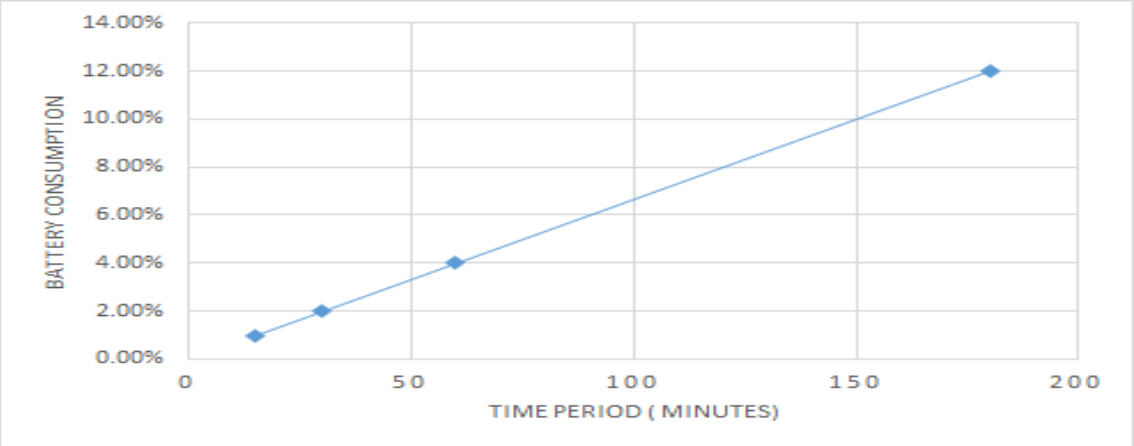


Figure 4.11: Battery Consumption (%)

From the result of the above Table 4.2, a linear regression diagram is presented to find the relationship between the battery consumption and time as shown in Figure 4.11; y represents the amount of battery consumed, and x represents the time of approximately 200 minutes. It can be seen from the above figure that there was an upward trend in the percentage of the battery consumption over the time between 15 and 180 minutes.

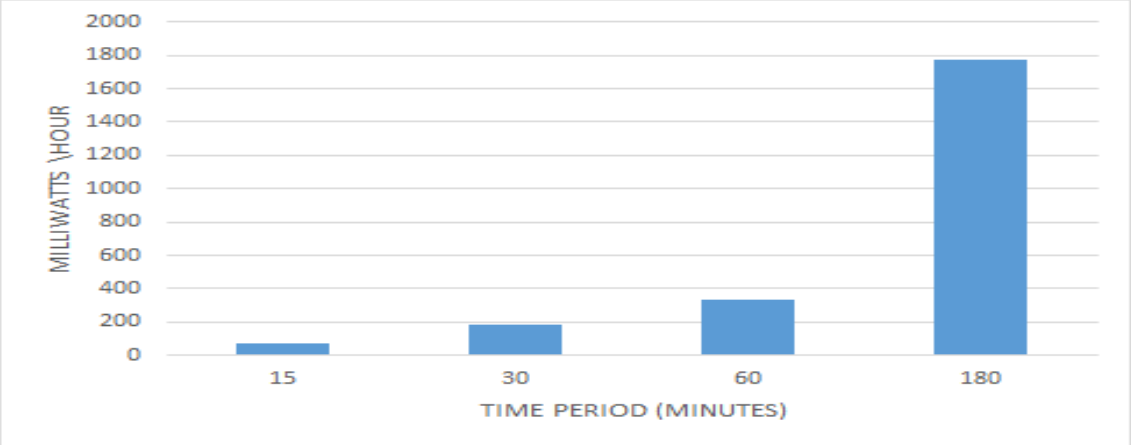


Figure 4.12: Power Consumption (mWh)

Besides, it can be seen in Figure 4.12 that the diagram demonstrates the amount of battery power in Milliwatt (mW)h in an hour over the periods of 180 minutes; y represents the amount of mWh, while x represents the time. The above figure illustrates that the power consumption shows a gradual increase between the times of 15, and 60 minutes from 100 mWh to almost 300 mWh. Then, it went up significantly to almost 1780 mWh during the testing periods of 18 minutes.

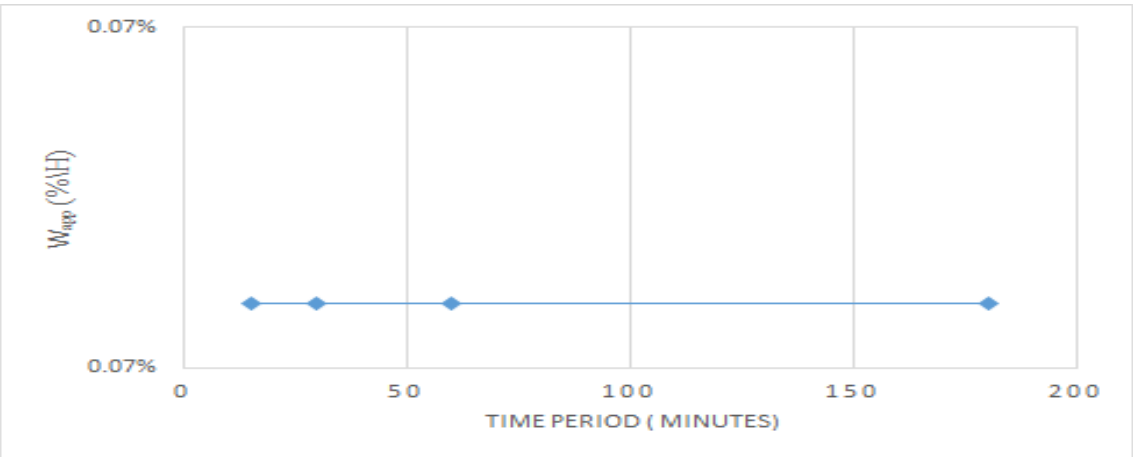


Figure 4.13: Weight of Application W_{app}

Additionally, Figure 4.13 shows a linear regression diagram that describes the relationship between the W_{app} over different time periods; y represents the W_{app}, and x

represents the time of almost 180 minutes. As can be seen in the above chart, the weight of the application during the testing periods between 15, and 180 minutes remained stable at 0.07%.

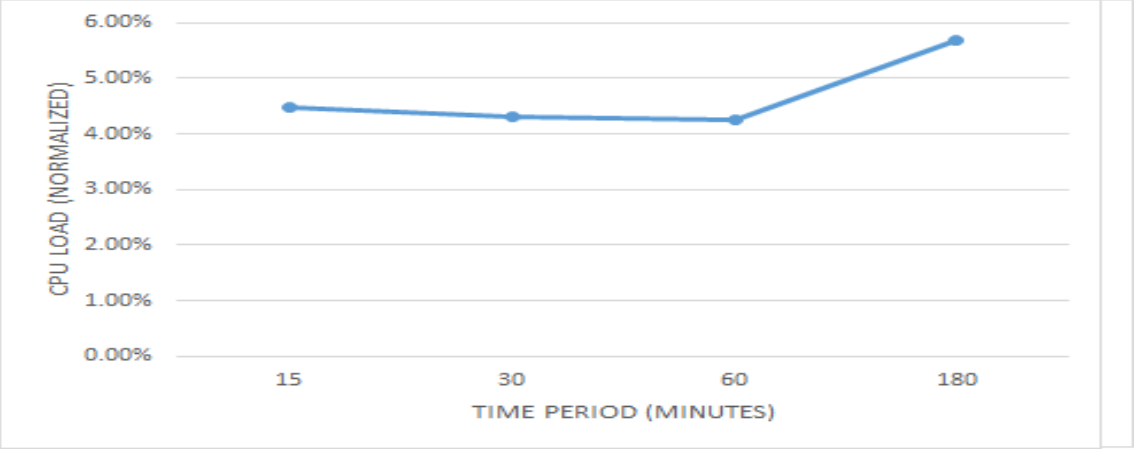


Figure 4.14: CPU Loads (Normalized)

Finally, Figure 4.14 shows a diagram for the CPU loads (Normalized) over 180 minutes. CPU loads (Normalized) reports the usage of the application with respect to the maximum potential of the CPU; for example, if a device has a single CPU core with a maximum frequency of 1000 MHz, this means in 1 second there are 1000 M clock cycles. If the application uses only 100 M of these cycles for work, CPU loads would show 10 % (100/1000). As is shown in the above graph, the application started to use 4.5% of CPU loads during the testing period of 15 minutes. Afterwords, over the times 30 and 60 minutes, the trend decreased slightly from 4.2% to almost 4.1%. It then rose dramatically during the testing period of 180 minutes.

The next chapter 5 will analyse the above findings.

4.4 Conclusion

The results of the research were collected, reported, and presented in a visual and graphical manner. There were three case studies that were used to simulate real life scenarios. The first case explained how the application could help the users to find common friends. The second case illustrated when two friends are in the same place, the application aids to inform them about nearby friends with the actual distance. In the last case the application helped the users to find people who share the same interest by suggesting a new friend who are in the surround environment. These cases were used to demonstrate the applicability of the study approach.

In addition, the performance of the developed application CommonFriends was evaluated using the formulas proposed by Ferreira et al. (2013). These formulas enabled the researcher to calculate the weight of the application, and the battery depletion rate. Another diagnostic application, suggested by Ferreira et al. (2015), was used to test the application during different periods: 15, 30, 60, and 180 minutes, allowing the researcher to measure the power, and the battery consumption of the proposed application.

In the following chapter, the results of the experiments, and findings will show a comparative analysis.

Chapter 5

Discussion

5.1 Introduction

The previous Chapter 4 reported the findings of the research according to the proposed architecture in section 3.3.2. The findings of the approach help to find one way to solve the research questions that are presented in Chapter 1. This Chapter consists of several sections: Section 5.2 gathers the findings of this research to answer the developed research questions. Section 5.3 compares the findings with previous studies, and analyses the graphs shown in Section 4.3. In addition, it discusses the usability of CommonFriends application to ensure the users' understanding of the application as well as evaluating the tools used in this research. It also explains how security and privacy are handled in the proposed framework. Finally, the conclusion section summarises the main points of this chapter.

5.2 Review of the Research Questions

The main objective of this section is to review the five research questions proposed in Chapter 1, and to explain the research approaches to these questions as the contribution

of this work.

Q1: How to model social context? Until now, there has been no commonly agreed definition, and conceptual model for social context.

This study has proposed an ontology-based model to find relationships on OSN: Facebook, Twitter, and LinkedIn, but the model was only implemented on the social context from Facebook, due to the time restriction of this research. This model can be extended in the future to be able to manage other types of social contexts, such as, user interaction, and events.

Q2: How to effectively acquire social context information from various sources like OSNs including Facebook, LinkedIn, and Twitter?

The *Allauth* package was used to authenticate the framework with the user's account in Facebook. Then, *Facepy* library was used to acquire social contexts from Facebook. However, the disadvantage of *Facepy* is that it can only access Facebook API because every OSN has defined their own APIs differently. Thus, it is recommended to use a package called *Urllib* already included with Python, where researchers can define their own code to access different OSNs, because it supports accessing remote resources with HTTP network protocol.

Q3: How to populate the social context model with actual context data?

The *RDFlib* package provides two techniques to manage the ontology by both the *Python* code, or the *SPARQL* query. In this study, Python code was used to populate the ontology with the social contexts, and SPARQL query was used to retrieve the social contexts from the ontology. However, the drawback of this package is that it does not include HermiT reasoner to validate the consistency of the ontology.

Q4: How to manage the social context model, and social context information so that it can be effectively used by software applications, and services?

The *RDFlib* package was used in this research, allowing the researcher to perform many tasks. For example, it supports different ontology formats, and includes the

implementation of SPARQL, the semantic query language for ontology. It is similar to SQL queries enabling the researcher to write procedure statements with get, insert, update, and delete. One the other hand, outsider applications send and receive contexts in JSON format through the framework's APIs.

Q5How to rapidly develop such software services so it can dynamically adapt to changes to the user's social context?

Having determined the types of the social contexts for the service, the framework can send requests directly to Facebook Graph in order to update any outdated information. This procedure is performed every time the users login to the framework from the mobile application. However, the shortcoming of this manner is in a case where the user does not log in and out more often, causing outdated information. Thus, it is suggested to use a tool called *Cron* that will enable the framework to schedule a task, for instance every three hours, to update the user's information.

5.3 Discussion

The purpose of this section is to highlight the major statistical findings from the results Chapter 4, and interpret them. Section 5.3.1 discusses the benefits of the approach applicability, and brings efficiency to social-context aware applications. Section 5.3.2 reviews the performance of the framework. Section 5.3.3 evaluates the usability of CommonFriends application that was developed to find common friends between users. Section 5.3.4 shows the advantages, and the disadvantages of the tools used in this approach.

5.3.1 Discussion of the Framework

As shown in Chapter 2, most studies defined social contexts based on their approaches to provide social aware services. However, after completing the objective of the study, and

understanding the meaning of social contexts, it can be referred to social information that the users give to describe themselves in the virtual communities. This definition is correlated with the second theory of social contexts, which integrates both physical, and virtual interactions mentioned in Chapter 2.

Based on the implementation of the proposed framework in section 3.3.2, it can be analysed that the centralized network architecture is more effective than P2P architecture for social aware applications on mobile devices. MobiClique used Bluetooth as a way of communication between two devices, where the application begins by detecting the nearby device. Then, the devices connect together to exchange the data. This limits the ability of the application because the devices cannot exchange data, if the users are not paired previously. In addition, Bluetooth is slow when it comes to exchanging data as the users have to stay within a particular distance in order to complete the process of exchanging the data (Pietiläinen et al., 2009). Unlike CommonFriends, it has been implemented in a different manner as it uses Bluetooth to detect the nearby device. Then, the detected device's MAC address is sent to the server to check whether they have something in common between them. Therefore, both users are not required to stay close to each other until the process is completed.

In addition, MobiClique processes the obtained data in the mobile application. This could affect the performance of the mobile device, which leads to consuming the device's storage and battery. In contrast, CommonFriends application uses a small amount of space 4.84 MB. It also does not perform heavy workloads on the mobile phone, which leads to consuming the CPU, and battery power.

In terms of the design of the ontology, SCIM middleware has a different view on modelling social contexts. It classifies the relationships in social contexts as people-centric and object-centric. People-centric is when a person identifies an individual as one of the family; for example, a father, while object-centric is when people participate in the same activities in OSN (Kabir et al., 2012). Although, this design could provide

a rich volume of social contexts, the social contexts can be difficult to obtain because the users then would be concerned about their privacy in the social aware application. Compared to this study, the model has been used only to acquire one type of relationship that involves the user's profile in Facebook. This relationship only consists of the public social context, which the users do not mind sharing with others.

However, there are many Ontology-based models that have been proposed for representing OSNs in the semantic web community, such as SCIM (Kabir et al., 2012), and SCOnto (Kabir et al., 2014). Some of these models considered the event interactions in the virtual communities, while other models recognized the relationships between users. Even though these models show a huge divergence in the design and the objective, they have a common perspective, which is obtaining social contexts from Facebook and other OSNs. In comparison to this study, the ontology model has been designed to analyse the user's friend lists on Facebook, Twitter, and Linkedin, but, this study has only implemented Facebook. However, combining all the above three models in one model could help in building a significant social aware service that covers most of the relationships, and social interactions.

5.3.2 Discussion of the Framework Performance

Although this experiment was performed on one mobile device, it has found some significant arguments. CommonFriends application was tested using QTP application, and the results obtained show a steady upward trend for battery consumption over the periods of 285 minutes. It can be analysed that CommonFriends consumes approximately 1% of battery power every 15 minutes; for example, if the application is running for 4 hours, it only consumes 16% of the battery. This indicates that CommonFriends uses minimal resources that is best for battery life in the mobile phone.

Another interesting point is that when Δ Battery, how much the battery is depleted,

was calculated during the testing sessions over the periods of 285 minutes. The result of the calculation was impressive: it amounts to 19% of battery consumption, which is similar to the result obtained from QTP. Similarly, BDR, how fast the battery is running out, was calculated using the formula suggested by Ferreira et al. (2013). The result is statistically significant that the application only consumes 4% per hour, which is also the same result as the one received from QTP. These results prove the stability of the application; in other words, the application can run almost the whole day without any noticeable battery consumption or unexpected performance.

In addition, the weight of the application CommonFriends, calculated using the formula proposed by Ferreira et al. (2013), showed constant results. During the test for the following periods: 15, 30, 60, and 180 minutes, the application weight remained constant at 0.07% indicating that the application performance is stable.

It is interesting that CommonFriends uses minimal resources of the CPU loads. The application was tested for the following periods: 15, 30, 60, and 180 minutes. During the times between 15 and 60 minutes, the application's usage of the CPU shows stable results of between 4% and 5%. However, when the CPU loads of the application were tested for 180 minutes, the usage of the CPU loads increased slightly to be around 5.5%. This reveals that the application does not overload the CPU and it only uses minimal resources.

Despite the fact that CommonFriends application shows stable results when it comes to battery consumption, the amount of power in mWh used during different periods shows unusual actions. It can be seen that, during the test of the application, the amount of power started in the first 15 minutes at 90 mWh. Then, it increased steadily for the following periods of times: 30 minutes by 180 mWh, and 60 minutes by almost 360 mWh. However, when the application was tested for 180 minutes, the amount of power used rose dramatically by around 1700 mWh, which is considered to be unexpected behaviour.

5.3.3 Discussion of the Usability of CommonFriends

As the CommonFriends application was completed, the researcher started a thorough testing process. First, the application was extensively tested with three different users using user test accounts provided by Facebook testing environment to determine the usability of the application, and to simulate real scenarios. All users were given a mobile device with a CommonFriends application already installed.

The first user found the application easy to use, but she suggested that each page of the application should be titled so that the user knows what page she is at. For example, after login to Facebook, the user is transferred to the home page where there are four options. When the user clicks on the option 'Account', the user does not know if they are in the Account page.

The second participant recommended that, after the user logs in to Facebook, the application should show a label with a welcome message, and the user's name. Thus, the user recognizes that the login process was successful.

The third user stated that when the application requests the user to turn on the Bluetooth, the user has the options: Accept or Decline. If the user declines the request, the application does not work, even if the user turns on the Bluetooth later from the mobile's setting. Hence, the researcher added an option to allow the user to turn the Bluetooth on or off within the application.

5.3.4 Discussion of the Tools

There are different tools that have been used to develop an artifact that solves the problem in this study. These tools are Python-Django, Django-Rest-Framework, Django-Allauth, RDFlib, Facepy, SQLite. The following Table 5.1 highlight the main features that were used in this research in order to implement the framework, and presents the findings.

Table 5.1: Features and Missing Features of Tools

Tool Name	Feature	Missing Feature
Python-Django	-Easy to install. -Flexible to change the initial code. - Well Documented -Used by popular companies.	
Django-Rest-Framework	- Easy to install. - Well Documented. - Provide different API methods.	- Difficult to change the design.
Django-Allauth	- Easy to install. - Support almost all OSNs.	- Does not provide an API for OSNs. - Not much information about the code.
RDFlib	- Easy to install. - Well Documented. - Difficult to use.	- Does not provide a Reasoning tool, such as HermiT Reasoner, to check the consistency of the ontology.
Facepy	- Easy to use.	- Only support Facebook.
SQLite	- No separate server for setup. - It is portable across all operating systems. - Support SQL syntax. - Ready to use with Django Framework.	- Database size restricted to 2GB. - Not very scalable. - It is slow.

5.3.5 Discussion of Security and Privacy

SCWS gathers social contexts from Facebook such as user profiles and user's friend lists, which are usually regarded as public information. However, users do not wish this information to be shared unless they approve that. Therefore, the security and privacy should be considered or the application will not be widely accepted. A context management system should perform at least one type of authentication methods. Due to the time limitation of this research, this framework does not implement access control in order to secure the access to the user's information as achieved in MobiSoC (Gupta et al., 2009), SCIMS (Kabir et al., 2012), VENETA (Von Arb et al., 2008), and Yarta

(Toninelli et al., 2011). However, several approaches to implement security and privacy can be proposed in future work.

On this approach, security and privacy were accomplished using API Open Authorization (OAuth). The selected criteria for this method were that, as the user logs in to Facebook through the application, the user token is acquired, which is then sent to the server for validation by connecting to Facebook API. Having validated the user token, the server sends the application a server token allowing the users to remotely access the framework.

5.4 Conclusion

Upon understanding the findings presented in Section 4, they were discussed in this chapter, helping to answer the research questions. Firstly, social context has been defined in this study according to the approach developed. This definition is related to the second theory, which involves physical, and virtual interactions in the OSNs.

Secondly, the proposed framework was compared with MobiClique middleware finding that the centralized architecture is more effective than P2P architecture in terms of connectivity. Similarly, a comparison between the proposed ontology model and other ontologies, SCIM, (Kabir et al., 2012) and SCOnto (Kabir et al., 2014) showed that combining these ontologies together could provide a comprehensive model covering the most important social contexts in the virtual communities.

Thirdly, the results obtain from QTP were analysed and compared with the formulas proposed by Ferreira et al. (2013). It was found that the CommonFriends application consumes only 4% per hour of battery and that the weight of the application remained stable at 7%, which indicates the steady performance of the application.

Fourthly, CommonFriends application was reviewed by three users to check the usability of the application to help improve the users' interactions. They suggested to

add a title for each page, show a welcome message with the user's name, and allow users to turn the Bluetooth on/off within the application. Afterwards, the tools used to develop the proposed architecture were evaluated by outlining the advantages and disadvantages.

Finally, users do not wish to share their information unless they approve it. Thus, security and privacy should be considered. This study did not use access control, it only implemented OAuth to prevent unauthorized access to the user's information.
The following chapter will summarise the whole research as well as offering suggestions for future work.

Chapter 6

Conclusion

6.1 Introduction

Chapter 1 explains the background and motivations of the research. Chapter 2 highlights the recent studies of social aware applications. These two chapters identify the importance of this research. Chapter 3 demonstrates the methodology of the research and illustrates the research design, the SCWS architecture proposed, and the concept of this architecture. Chapter 4 presents the findings of the research. Chapter 5 answers the research questions and discusses the most notable findings.

This Chapter concludes this thesis. It has proposed an architecture and implementation to test its capability of dealing with social contexts acquired from Facebook. There are huge volumes of social contexts that can acquired from OSNs; therefore, understanding social contexts can help the researchers to design a reliable model that is able to cover most of them. It also assists the researchers to build socially aware applications as fast as possible that can provide useful information.

This Chapter reviews the most significant aspects of the findings. It is organized into several sections; Section 6.2 summarises the findings of the research; Section 6.3 suggests recommendations of the approach of this study; Section 6.4 presents the

limitation of the research; Section 6.5 suggests some ideas for future work.

6.2 Summary of the Research Results

The main objective of this research is to build a social aware application that is able to find relationships on Facebook. Upon understanding the social contexts, an ontology-based model was designed that is capable of storing user's information and user's friend lists and making them readable for machines. Having designed the model, it was compared with several studies that used Facebook's social contexts: SCIM (Kabir et al., 2012), and SCOnto (Kabir et al., 2014). It was found that the design of the ontology of SCIM focuses on the relationships of the users, categorized into people-centric relationships and object-centric relationships. People-centric relationships exist where people identify a member of their family, while object-centric relationships exist where people are engaged in the same activities in the OSNs. SCOnto attempted to understand the social interaction in the OSNs. However, the proposed model in this study focuses on building an ontology aiming to find common friends between users on Facebook.

The proposed SCWS architecture consists of several layers that can acquire contexts from Facebook API, store the contexts into the ontology, infer the contexts, and provide the results to outsider applications through API. In addition, the framework, based on a centralized architecture, was compared with P2P architecture. It has been found that the centralized architecture is more reliable in terms of the users' not being required to be near each other during the exchange of the information and that it puts a huge pressure on the mobile's CPU. This may lead to an effect on the user's interactions because, in P2P architecture, most of the process is performed in the mobile.

After completing the above steps, three use cases were demonstrated to simulate real scenarios. The first scenario explains that two people work together, but they do not know whether they have common friends in Facebook. The second scenario helps

the user to know if their friends are nearby. The last scenario allows the user to find a new friend who shares the same interests.

Afterwards, QTP tool, suggested by Ferreira et al. (2015), was used to evaluate CommonFriends application for four different periods: 15, 30, 60, and 180 minutes. Each period was performed three time to ensure the accuracy of the results, which are then calculated using the equations proposed by Ferreira et al. (2013). The calculation results show that the weight of the application remained stable at 0.07% and that the application consumes minimal resources of CPU by almost 5%, confirming the reliability of the application performance.

6.3 Recommendations

Bluetooth RSSI was used in this study to measure the distance between devices; however, sometimes it does not show accurate results. It is, therefore, recommended to use GPS to calculate the distance among devices to provide more precise results.

Although, the researcher used the RDFlib tool to manage the ontology model, it does not provide a reasoner tool to check the consistency of the ontology, such as HermiT Reasoner, which may cause problems in the future. However, Jena, a powerful Java tool, can manage an ontology-based model smoothly and it can be integrated with Python using some available library, such as Jython acting as a bridge between Java and Python. Hence, allowing Jena to work in the Python environment will result in a robust tool to manage the ontology model.

6.4 Research Limitations

The ontology was designed to manage social contexts from several OSNs, such as Facebook, Twitter, and LinkedIn. However, because of the time limitation of this

research, Twitter, and LinkedIn could not be implemented. Therefore, they should be involved in the future to confirm the ontology model.

Another limitation was that there are different mobile with differentOSs, namely, Apple IOS, and Microsoft Windows Phone. This study only developed an application for Android OS; therefore, the performance, and the functionality may not be applicable for other platforms as different OSs have different concepts when it comes to the user usability and application performance.

In addition, Facebook does not permit the developers to acquire user's friend lists, unless they provide reasons for using this information, because Facebook considered this data as private. Nevertheless, the researcher obtained the user's friend lists using another API called `taggable friends`, allowing the developers access to the user's name, and profile picture. However, the user's emails and Facebook ID could be accessed, which may cause a conflict in the future because, when the framework acquires social contexts from Facebook, the user's friend are identified in the ontology by their full names. If one of the user's friends changes his\her name, the framework will consider them as a new friend and store them.

6.5 Future Work

Due to the time limits of this research, this framework lacks a number of functionalities. The first is security and privacy, as this framework is only implemented by the OAuth method. However, this can be improved in the future by applying a semantic access control to the ontology, helping to mange user access to the information stored.

The current approach did not include a reasoner or machine-learning techniques. However, in order to enhance the framework, these components should be involved in the future. This will aid in classifying the ontology, which can handle complexity and the accuracy of knowledge.

Another important point to consider is that this approach only considered social contexts obtained from Facebook; thus, in the future it is suggested that the framework should handle other OSNs such as Twitter, and LinkedIn because they contain huge volumes of social contexts that can be used to provide a comprehensive knowledge-based service.

Finally, in this study, one ontology file was used to store all the social contexts, which may give rise a large concerns in the future when many users are using the service. Hence, it would be ideal if the social contexts for each user are stored in a separate ontology file by adjusting the framework as follows: First, the server acquires social contexts from OSNs and populates the ontology with the contexts. Second, when a nearby device is detected, the server sends a copy of the user's ontology to either or both devices. Lastly, the mobile phone starts retrieving the social contexts and compares them to find mutual friends. This would help to alleviate the workload on the server by sharing the work among the mobile devices.

References

Abowd, G. D., Atkeson, C. G., Hong, J., Long, S., Kooper, R. & Pinkerton, M. (1997). Cyberguide: A mobile context-aware tour guide. *Wireless networks*, *3*(5), 421–433.

Abowd, G. D., Dey, A. K., Brown, P. J., Davies, N., Smith, M. & Steggles, P. (1999). Towards a better understanding of context and context-awareness. In *Proceedings of the 1st international symposium on handheld and ubiquitous computing* (pp. 304–307). London, UK, UK: Springer-Verlag. Retrieved from `http://dl.acm.org/citation.cfm?id=647985.743843`

Arnaboldi, V., Conti, M. & Delmastro, F. (2014). Cameo: A novel context-aware middleware for opportunistic mobile social networks. *Pervasive and Mobile Computing*, *11*, 148–167.

Bachrach, J. & Taylor, C. (2005). Localization in sensor networks. *Handbook of sensor networks: Algorithms and Architectures*, *1*.

Backhaus, G. & Murungi, J. (2004). *Earth ways: Framing geographical meanings*. Lexington Books.

Baldauf, M., Dustdar, S. & Rosenberg, F. (2007). A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, *2*(4), 263–277.

Beach, A., Gartrell, M., Akkala, S., Elston, J., Kelley, J., Nishimoto, K., ... others (2008). Whozthat? evolving an ecosystem for context-aware mobile social networks. *Network, IEEE*, *22*(4), 50–55.

Beach, A., Gartrell, M., Xing, X., Han, R., Lv, Q., Mishra, S. & Seada, K. (2010). Fusing mobile, sensor, and social data to fully enable context-aware computing. In *Proceedings of the eleventh workshop on mobile computing systems & applications* (pp. 60–65).

Bennaceur, A., Singh, P., Raverdy, P.-G. & Issarny, V. (2009). The ibicoop middleware: Enablers and services for emerging pervasive computing environments. In *Pervasive computing and communications, 2009. percom 2009. ieee international conference on* (pp. 1–6).

Bettini, C., Brdiczka, O., Henricksen, K., Indulska, J., Nicklas, D., Ranganathan, A. & Riboni, D. (2010). A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, *6*(2), 161–180.

Biamino, G. (2011, March). Modeling social contexts for pervasive computing environments. In *Pervasive computing and communications workshops (percom workshops), 2011 ieee international conference on* (p. 415-420). doi:

10.1109/PERCOMW.2011.5766925

Boldrini, C., Conti, M., Delmastro, F. & Passarella, A. (2010). Context-and social-aware middleware for opportunistic networks. *Journal of Network and Computer Applications*, *33*(5), 525–541.

Boldrini, C., Conti, M., Jacopini, J. & Passarella, A. (2007). Hibop: a history based routing protocol for opportunistic networks. In *World of wireless, mobile and multimedia networks, 2007. wowmom 2007. ieee international symposium on a* (pp. 1–12).

Bottazzi, D., Montanari, R. & Toninelli, A. (2007). Context-aware middleware for anytime, anywhere social networks. *Intelligent Systems, IEEE*, *22*(5), 23–32.

Byun, J. & Park, S. (2011). Development of a self-adapting intelligent system for building energy saving and context-aware smart services. *Consumer Electronics, IEEE Transactions on*, *57*(1), 90–98.

Carminati, B., Ferrari, E., Heatherly, R., Kantarcioglu, M. & Thuraisingham, B. (2009). A semantic web based framework for social network access control. In *Proceedings of the 14th acm symposium on access control models and technologies* (pp. 177–186).

Carminati, B., Ferrari, E. & Perego, A. (2006). Rule-based access control for social networks. In *On the move to meaningful internet systems 2006: Otm 2006 workshops* (pp. 1734–1744).

Carrera Carbó, C. (2012). Development of an android apk for bluetooth localization.

Cass, S. (2015). The 2015 top ten programming languages. *IEEE Spectrum, July*, *20*.

Chen, C.-M. & Li, Y.-L. (2010). Personalised context-aware ubiquitous learning system for supporting effective english vocabulary learning. *Interactive Learning Environments*, *18*(4), 341–364.

Cheverst, K., Davies, N., Mitchell, K., Friday, A. & Efstratiou, C. (2000). Developing a context-aware electronic tourist guide: Some issues and experiences. In *Proceedings of the sigchi conference on human factors in computing systems* (pp. 17–24). New York, NY, USA: ACM. Retrieved from `http://doi.acm.org/10.1145/332040.332047` doi: 10.1145/332040.332047

Choi, C., Park, I., Hyun, S. J., Lee, D. & Sim, D. H. (2008). Mire: A minimal rule engine for context-aware mobile devices. In *Digital information management, 2008. icdim 2008. third international conference on* (pp. 172–177).

Cristofaro, E. D., Durussel, A. & Aad, I. (2011). Reclaiming privacy for smartphone applications. In *Pervasive computing and communications (percom), 2011 ieee international conference on* (pp. 84–92).

Dey, A. K. (2001, January). Understanding and using context. *Personal Ubiquitous Comput.*, *5*(1), 4–7. Retrieved from `http://dx.doi.org/10.1007/s007790170019` doi: 10.1007/s007790170019

Dey, A. K., Abowd, G. D. & Salber, D. (2001). A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-computer interaction*, *16*(2), 97–166.

Drakatos, S., Pissinou, N., Makki, K. & Douligeris, C. (2007). A context-aware cache structure for mobile computing environments. *Journal of Systems and Software*,

80(7), 1102–1119.

Duggan, M. & Page, D. (2015). Mobile messaging and social media 2015. *Pew Research Center: http://www. pewinternet. org/2015/08/19/mobile-messaging-and-social-media-2015*.

Ellison, N. B. et al. (2007). Social network sites: Definition, history, and scholarship. *Journal of Computer-Mediated Communication*, *13*(1), 210–230.

Eugster, P. T., Garbinato, B. & Holzer, A. (2009). Middleware support for context-aware applications. In *Middleware for network eccentric and mobile applications* (pp. 305–322). Springer.

Facebook. (2016). *Api* (Tech. Rep.). Retrieved from `https://developers.facebook.com/`

Fagin, R., Halpern, J. Y. & Megiddo, N. (1990). A logic for reasoning about probabilities. *Information and computation*, *87*(1), 78–128.

Ferreira, D., Ferreira, E., Goncalves, J., Kostakos, V. & Dey, A. K. (2013). Revisiting human-battery interaction with an interactive battery interface. In *Proceedings of the 2013 acm international joint conference on pervasive and ubiquitous computing* (pp. 563–572).

Ferreira, D., Kostakos, V. & Dey, A. K. (2015). Aware: mobile context instrumentation framework. *Frontiers in ICT*, *2*, 6.

Flentge, F., Weber, S. G., Behring, A. & Ziegert, T. (2008). Designing context-aware hci for collaborative emergency management. In *Int'l workshop on hci for emergencies in conjunction with chi* (Vol. 8, p. 2008).

Forcier, J., Bissex, P. & Chun, W. J. (2008). *Python web development with django*. Addison-Wesley Professional.

Gámez, N., Cubo, J., Fuentes, L. & Pimentel, E. (2012). Configuring a context-aware middleware for wireless sensor networks. *Sensors*, *12*(7), 8544–8570.

Geerts, G. L. (2011). A design science research methodology and its application to accounting information systems research. *International Journal of Accounting Information Systems*, *12*(2), 142–151.

Gennari, J. H., Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubézy, M., Eriksson, H., ... Tu, S. W. (2003). The evolution of protégé: an environment for knowledge-based systems development. *International Journal of Human-computer studies*, *58*(1), 89–123.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge acquisition*, *5*(2), 199–220.

Gu, T., Pung, H. K. & Zhang, D. Q. (2004). Toward an osgi-based infrastructure for context-aware applications. *Pervasive Computing, IEEE*, *3*(4), 66–74.

Gu, T., Pung, H. K. & Zhang, D. Q. (2005). A service-oriented middleware for building context-aware services. *Journal of Network and computer applications*, *28*(1), 1–18.

Gupta, A., Kalra, A., Boston, D. & Borcea, C. (2009). Mobisoc: a middleware for mobile social computing applications. *Mobile Networks and Applications*, *14*(1), 35–52.

Halpin, T. & Morgan, T. (2010). *Information modeling and relational databases*. Morgan Kaufmann.

Han, L., Jyri, S., Ma, J. & Yu, K. (2008). Research on context-aware mobile computing. In *Advanced information networking and applications-workshops, 2008. ainaw 2008. 22nd international conference on* (pp. 24–30).

Haythornthwaite, C. (2005). Social networks and internet connectivity effects. *Information, Community & Society*, *8*(2), 125–147.

Henricksen, K. & Indulska, J. (2006). Developing context-aware pervasive computing applications: Models and approach. *Pervasive and mobile computing*, *2*(1), 37–64.

Henricksen, K., Livingstone, S. & Indulska, J. (2004). Towards a hybrid approach to context modelling, reasoning and interoperation. In *Proceedings of the first international workshop on advanced context modelling, reasoning and management, in conjunction with ubicomp* (Vol. 2004).

Iamnitchi, A., Blackburn, J. & Kourtellis, N. (2012). The social hourglass: An infrastructure for socially aware applications and services. *IEEE Internet Computing*, *16*(3), 13–23.

Indulska, J., Robinson, R., Rakotonirainy, A. & Henricksen, K. (2003). Experiences in using cc/pp in context-aware systems. In *Mobile data management* (pp. 247–261).

Jung, J., Kang, D. & Bae, C. (2013). Distance estimation of smart device using bluetooth. *Personal Computing Platform Research Team*, 13–18.

Kabir, M. A. (2013, March). Modeling, managing and reasoning about social contexts for socially-aware applications. In *Pervasive computing and communications workshops (percom workshops), 2013 ieee international conference on* (p. 419-420). doi: 10.1109/PerComW.2013.6529531

Kabir, M. A., Han, J., Yu, J. & Colman, A. (2012). Scims: a social context information management system for socially-aware applications. In *Advanced information systems engineering* (pp. 301–317).

Kabir, M. A., Han, J., Yu, J. & Colman, A. (2014). Inferring user situations from interaction events in social media. *The Computer Journal*. Retrieved from `http://comjnl.oxfordjournals.org/content/early/2014/11/23/comjnl.bxu131.abstract` doi: 10.1093/comjnl/bxu131

Kaplan, A. M. & Haenlein, M. (2010). Users of the world, unite! the challenges and opportunities of social media. *Business horizons*, *53*(1), 59–68.

Kayes, I. & Iamnitchi, A. (2013). Aegis: A semantic implementation of privacy as contextual integrity in social ecosystems. In *Privacy, security and trust (pst), 2013 eleventh annual international conference on* (pp. 88–97).

Kietzmann, J. H., Hermkens, K., McCarthy, I. P. & Silvestre, B. S. (2011). Social media? get serious! understanding the functional building blocks of social media. *Business horizons*, *54*(3), 241–251.

Klyne, G. & Carroll, J. J. (2006). Resource description framework (rdf): Concepts and abstract syntax.

Kourtellis, N. (2012). On the design of socially-aware distributed systems.

Kourtellis, N., Finnis, J., Anderson, P., Blackburn, J., Borcea, C. & Iamnitchi, A. (2010). Prometheus: User-controlled p2p social data management for socially-aware applications. In *Proceedings of the acm/ifip/usenix 11th international conference on middleware* (pp. 212–231).

Kulkarni, D. & Tripathi, A. (2008). Context-aware role-based access control in pervasive computing systems. In *Proceedings of the 13th acm symposium on access control models and technologies* (pp. 113–122).

Levy, J. (2010). *Facebook marketing: Designing your next marketing campaign*. Pearson Education.

Li, J. & Dabek, F. (2006). F2f: Reliable storage in open networks. In *Iptps*.

Liang, G. & Cao, J. (2015). Social context-aware middleware: A survey. *Pervasive and Mobile Computing*, *17, Part B*, 207 - 219. Retrieved from `http://www.sciencedirect.com/science/article/pii/S1574119214001916` (10 years of Pervasive Computing' In Honor of Chatschik Bisdikian) doi: http://dx.doi.org/10.1016/j.pmcj.2014.12.003

Liang, G., Cao, J. & Zhu, W. (2013). Circlesense: A pervasive computing system for recognizing social activities. In *Pervasive computing and communications (percom), 2013 ieee international conference on* (pp. 201–206).

Liao, L., Patterson, D. J., Fox, D. & Kautz, H. (2007). Learning and inferring transportation routines. *Artificial Intelligence*, *171*(5), 311–331.

LinkedIn. (2016a). *Api* (Tech. Rep.). Retrieved from `https://developer.linkedin.com/?u=0`

LinkedIn. (2016b). *Worldwide membership* (Tech. Rep.). Retrieved from `https://press.linkedin.com/about-linkedin`

Lu, J., Ma, L., Zhang, L., Brunner, J.-S., Wang, C., Pan, Y. & Yu, Y. (2007). Sor: a practical system for ontology storage, reasoning and search. In *Proceedings of the 33rd international conference on very large data bases* (pp. 1402–1405).

Lü, L. & Zhou, T. (2011). Link prediction in complex networks: A survey. *Physica A: Statistical Mechanics and its Applications*, *390*(6), 1150–1170.

Ma, C., Cao, J., Yang, L., Ma, J. & He, Y. (2014). Effective social relationship measurement based on user trajectory analysis. *Journal of Ambient Intelligence and Humanized Computing*, *5*(1), 39–50.

Masse, M. (2011). *Rest api design rulebook*. " O'Reilly Media, Inc.".

Mislove, A., Gummadi, K. P. & Druschel, P. (2006). Exploiting social networks for internet search. In *5th workshop on hot topics in networks (hotnets06). citeseer* (p. 79).

Mokhtar, S. B., McNamara, L. & Capra, L. (2009). A middleware service for pervasive social networking. In *Proceedings of the international workshop on middleware for pervasive mobile and embedded computing* (p. 2).

Nunamaker Jr, J. F., Chen, M. & Purdin, T. D. (1990). Systems development in information systems research. *Journal of management information systems*, *7*(3), 89–106.

Park, D. J., Hwang, S. H., Kim, A. R. & Chang, B. M. (2007, Nov). A context-aware smart tourist guide application for an old palace. In *Convergence information technology, 2007. international conference on* (p. 89-94). doi: 10.1109/ICCIT.2007.211

Paul-Stueve, T. & Wachsmuth, S. (2012, March). Towards a social context model and architecture for large-scale pervasive environments. In *Pervasive computing and communications workshops (percom workshops), 2012 ieee international conference on* (p. 619-624). doi: 10.1109/PerComW.2012.6197589

Peffers, K., Tuunanen, T., Rothenberger, M. A. & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of management information systems*, *24*(3), 45–77.

Pelusi, L., Passarella, A. & Conti, M. (2006). Opportunistic networking: data forwarding in disconnected mobile ad hoc networks. *Communications Magazine, IEEE*, *44*(11), 134–141.

Pietiläinen, A.-K., Oliver, E., LeBrun, J., Varghese, G. & Diot, C. (2009). Mobiclique: middleware for mobile social networking. In *Proceedings of the 2nd acm workshop on online social networks* (pp. 49–54).

Riboni, D. & Bettini, C. (2011). Owl 2 modeling and reasoning with complex human activities. *Pervasive and Mobile Computing*, *7*(3), 379–395.

Rowstron, A. & Druschel, P. (2001). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001* (pp. 329–350).

Ryan, N., Pascoe, J. & Morse, D. (1999). Enhanced reality fieldwork: the context aware archaeological assistant. *Bar International Series*, *750*, 269–274.

Schilit, B., Adams, N. & Want, R. (1994, Dec). Context-aware computing applications. In *Mobile computing systems and applications, 1994. wmcsa 1994. first workshop on* (p. 85-90). doi: 10.1109/WMCSA.1994.16

Schilit, B. N. & Theimer, M. M. (1994). Disseminating active map information to mobile hosts. *Network, IEEE*, *8*(5), 22–32.

Serral, E., Valderas, P. & Pelechano, V. (2010). Towards the model driven development of context-aware pervasive systems. *Pervasive and Mobile Computing*, *6*(2), 254 - 280. Retrieved from `http://www.sciencedirect.com/science/article/pii/S1574119209000613` (Context Modelling, Reasoning and Management) doi: http://dx.doi.org/10.1016/j.pmcj.2009.07.006

Shafer, G. et al. (1976). *A mathematical theory of evidence* (Vol. 1). Princeton university press Princeton.

Shaw, R., Troncy, R. & Hardman, L. (2009). Lode: Linking open descriptions of events. In *Asian semantic web conference* (pp. 153–167).

Sheng, Q. Z. & Benatallah, B. (2005). Contextuml: a uml-based modeling language for model-driven development of context-aware web services. In *Mobile business, 2005. icmb 2005. international conference on* (pp. 206–212).

Social. (2016). *Oxford english dictionary online* (Tech. Rep.). Retrieved from `http://www.oxforddictionaries.com/definition/english/social`

Social Network. (2016). *Cambridge english dictionary online* (Tech. Rep.). Retrieved from `http://dictionary.cambridge.org/dictionary/`

english/social-network

Solanas, A., Patsakis, C., Conti, M., Vlachos, I., Ramos, V., Falcone, F., ... others (2014). Smart health: a context-aware health paradigm within smart cities. *Communications Magazine, IEEE*, *52*(8), 74–81.

SQLite. (2016). *About sqlite* (Tech. Rep.). Retrieved from https://www.sqlite.org/about.html/

Statista. (2016, January). *Leading social networks worldwide as of january 2016, ranked by number of active users (in millions)* (Tech. Rep.). Retrieved from http://www.statista.com/statistics/272014/global-social-networks-ranked-by-number-of-users/

Sumi, Y., Etani, T., Fels, S., Simonet, N., Kobayashi, K. & Mase, K. (1998). C-map: Building a context-aware mobile assistant for exhibition tours. In *Community computing and support systems* (pp. 137–154). Springer.

Teles, A., Pinheiro, D., Gonçalves, J., Batista, R., Silva, F., Pinheiro, V., ... Endler, M. (2012). Mobilehealthnet: A middleware for mobile social networks in m-health. In *Proceedings of the 3rd international conference on wireless mobile communication and healthcare, mobihealth* (Vol. 12).

Toninelli, A., Khushraj, D., Lassila, O. & Montanari, R. (2008). Towards socially aware mobile phones. In *7th international semantic web conference* (pp. 68–75).

Toninelli, A., Pathak, A. & Issarny, V. (2011). Yarta: a middleware for managing mobile social ecosystems. In *Advances in grid and pervasive computing* (pp. 209–220). Springer.

Van Hage, W. R., Malaisé, V., Segers, R., Hollink, L. & Schreiber, G. (2011). Design and use of the simple event model (sem). *Web Semantics: Science, Services and Agents on the World Wide Web*, *9*(2), 128–136.

Van Kranenburg, H., Bargh, M. S., Iacob, S. & Peddemors, A. (2006). A context management framework for supporting context-aware distributed applications. *Communications Magazine, IEEE*, *44*(8), 67–74.

Von Arb, M., Bader, M., Kuhn, M. & Wattenhofer, R. (2008). Veneta: Serverless friend-of-friend detection in mobile social networking. In *Networking and communications, 2008. wimob'08. ieee international conference on wireless and mobile computing,* (pp. 184–189).

Wang, G., Gallagher, A., Luo, J. & Forsyth, D. (2010). Seeing people in social context: Recognizing people and social relationships. In *Computer vision–eccv 2010* (pp. 169–182). Springer.

Wang, X. H., Zhang, D. Q., Gu, T. & Pung, H. K. (2004, March). Ontology based context modeling and reasoning using owl. In *Pervasive computing and communications workshops, 2004. proceedings of the second ieee annual conference on* (p. 18-22). doi: 10.1109/PERCOMW.2004.1276898

Want, R., Hopper, A., Falcao, V. & Gibbons, J. (1992). The active badge location system. *ACM Transactions on Information Systems (TOIS)*, *10*(1), 91–102.

Wood, A. D., Stankovic, J. A., Virone, G., Selavo, L., He, Z., Cao, Q., ... Stoleru, R. (2008). Context-aware wireless sensor networks for assisted living and residential monitoring. *Network, IEEE*, *22*(4), 26–33.

Xing, B., Gronowski, K., Radia, N., Svensson, M. & Ton, A. (2011, March). Pocketsocial: Your distributed social context now in your pocket. In *Pervasive computing and communications workshops (percom workshops), 2011 ieee international conference on* (p. 322-324). doi: 10.1109/PERCOMW.2011.5766895

Yu, J., Sheng, Q. Z., Swee, J. K., Han, J., Liu, C. & Noor, T. H. (2015). Model-driven development of adaptive web service processes with aspects and rules. *Journal of Computer and System Sciences*, *81*(3), 533 - 552. Retrieved from `http://www.sciencedirect.com/science/article/pii/S0022000014001494` (Special Issue on selected papers from the 4th International Conference on Ambient Systems, Networks and Technologies (ANT 2013)) doi: http://dx.doi.org/10.1016/j.jcss.2014.11.008

Yu, X., Pan, A., Tang, L.-A., Li, Z. & Han, J. (2011). Geo-friends recommendation in gps-based cyber-physical social network. In *Advances in social networks analysis and mining (asonam), 2011 international conference on* (pp. 361–368).

Zhang, G. & Parashar, M. (2004). Context-aware dynamic access control for pervasive applications. In *Proceedings of the communication networks and distributed systems modeling and simulation conference* (pp. 21–30).

Zheng, Y. & Yano, Y. (2007). A framework of context-awareness support for peer recommendation in the e-learning context. *British Journal of Educational Technology*, *38*(2), 197–210.

Zhuang, J., Mei, T., Hoi, S. C., Hua, X.-S. & Li, S. (2011). Modeling social strength in social media community via kernel-based learning. In *Proceedings of the 19th acm international conference on multimedia* (pp. 113–122).

Appendix A

Framework Setup

A.1 Virtual Environment

Virtual Environment is a tool to keep all dependencies in a separate place. This helps to solve the issues of managing different versions for different projects.

1. To install virtualenv via *pip*:

   `pip install virtualenv`

2. To go to the file where the project is:

   `cd myproject`

3. To create a virtual environment:

   `virtualenv -p /usr/bin/python2.7 venv`

   where `venv` is the name of the environment. `python2.7` is the version of Python language Version.

4. To begin using the virtual environment:

   `source venv/bin/activate`.

5. To install packages as usual:

   `pip install django`

6. To deactivate the virtual environment:

   `deactivate`

A.2 Project Setup

1. Let's start building the project.

```
# Create the project directory
mkdir myproject
cd myproject

# Activate the created virtualenv
source env/bin/activate

# Install Django and Django REST framework into the virtualenv
pip install django
pip install djangorestframework

# Set up a new project with a single application
django-admin.py startproject socialawareness .  # Note the trailing
    ↪ '.' character
cd socialawareness
django-admin.py startapp commonfriends
django-admin.py startapp facebookapi
cd ..
```

2. Open up the app's directory to ensure you have the right files as shown in Figure A.1.

Figure A.1: App's Directory

3. Install the following packages into the created virtualenv:

```
# Install Facepy
pip install facepy

# Install Allauth
pip install django-allauth

# Install RDFlib
pip install rdflib
```

4. Configure `settings.py`, and add the packages installed in the previous steps to `INSTALLED_APPS` so it looks like:

```
INSTALLED_APPS = [
'django.contrib.admin',
'django.contrib.auth',
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
'rest_framework',
'rest_framework.authtoken',
# The Django sites framework is required
'rdflib',
'django.contrib.sites',
'allauth',
'allauth.account',
'allauth.socialaccount',
# Login Facebook provider
'allauth.socialaccount.providers.facebook',
'commonfriends',
'facebookapi',
]
```

5. Add `logging` setting as follows;

Listing A.1: Logging Errors

```
#Loggin Errors

LOGGING = {
        'version': 1,
        'disable_existing_loggers': False,
        'formatters': {
        'standard': {
        'format' : "[%(asctime)s] %(levelname)s [%(name)s:%(lineno)s]
            ↪ %(message)s",
        'datefmt' : "%d/%b/%Y %H:%M:%S"
        },
        },
        'handlers': {
        'null': {
```

```
        'level':'DEBUG',
        'class':'logging.NullHandler',
        },
        'logfile': {
        'level':'ERROR',
        'class':'logging.handlers.RotatingFileHandler',
        'filename': BASE_DIR + "/logfile",
        'maxBytes': 50000,
        'backupCount': 2,
        'formatter': 'standard',
        },
        'console':{
        'level':'DEBUG',
        'class':'logging.StreamHandler',
        'formatter': 'standard'
        },
        },
        'loggers': {
        'django': {
        'handlers':['console'],
        'propagate': True,
        'level':'WARN',
        },
        'django.db.backends': {
        'handlers': ['console'],
        'level': 'INFO',
        'propagate': False,
        },
        'commonfriends': {
        'handlers': ['console','logfile'],
        'level': 'DEBUG',
        },
        'facebookapi': {
        'handlers': ['console','logfile'],
        'level': 'DEBUG',
        },
        }
}
```

6. Add `Authentication` setting:

Listing A.2: Token Generation

```
REST_FRAMEWORK = {
        'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAuthenticated',
        ),
        'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework.authentication.SessionAuthentication',
        'rest_framework.authentication.TokenAuthentication',
        )
}
```

7. Finally, Add `Allauth` setting, and we save the file.

Listing A.3: Allauth

```
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
# auth and allauth settings
LOGIN_REDIRECT_URL = '/accounts/social/connections/'
ACCOUNT_LOGOUT_REDIRECT_URL = '/accounts/login'
SOCIALACCOUNT_QUERY_EMAIL = True
ACCOUNT_CONFIRM_EMAIL_ON_GET =True
ACCOUNT_EMAIL_VERIFICATION = 'optional'
SOCIALACCOUNT_PROVIDERS = \
        {'facebook':
        {'METHOD': 'oauth2',
        'SCOPE': ['email', 'public_profile', 'user_friends'],
        'AUTH_PARAMS': {'auth_type': 'reauthenticate'},
        'FIELDS': [
        'id',
        'email',
        'name',
        'first_name',
        'last_name',
        'verified',
        'locale',
        'timezone',
        'link',
        'gender',
        'updated_time'],
        'EXCHANGE_TOKEN': True,
        'VERIFIED_EMAIL': False,
        'VERSION': 'v2.4'
}
}
SITE_ID = 2
```

8. Sync our application with the database:

   `python manage.py migrate`

9. Create a superuser for the admin:

   `python manage.py createsuperuser`

10. Run the server

    `python manage.py runserver`

11. Navigate to `http://127.0.0.1:8000/admin` and login with the super user.
12. Change the *Domain Name* and *Display Name* under `Sites` to `http://127.0.0.1:8000/a` shown .

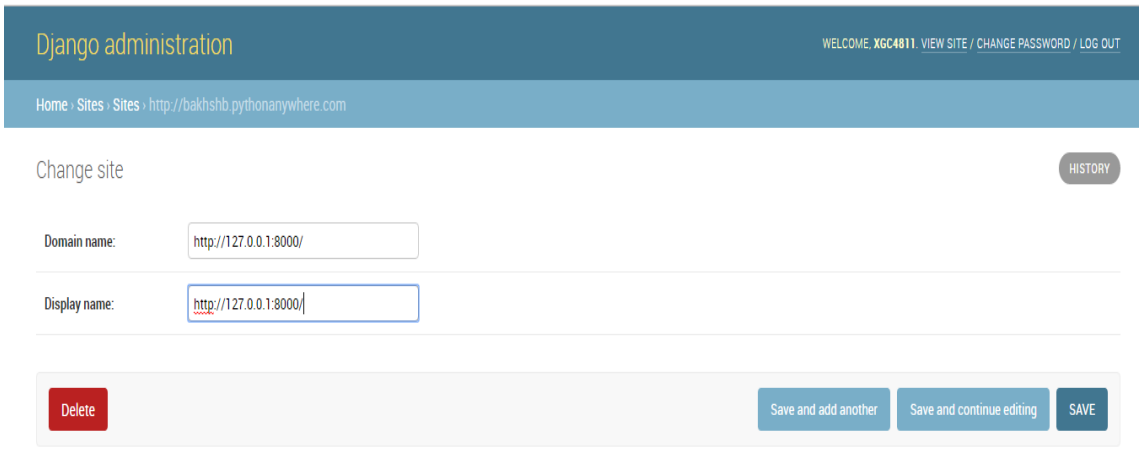


Figure A.2: Admin/Sites

13. Everything is setup but we need to do a few changes to the framework in order to acquire Facebook data, and store them in the ontology.
14. Navigate to the folder `commonfriends` and create a file named `helper.py`. The following code contains two class: Facebook Manager, Ontology Manager. The first class Facebook Manager gets the user's token and connect to Facebook's API to acquire their social context. The second class Ontology Manager has the following functions: create a new user, link the user with friends, search ontology with MAC address, find common friends.

Listing A.4: commonfriends/helper.py

```
from django.core.exceptions import ObjectDoesNotExist
from allauth.socialaccount.models import SocialToken, SocialApp,
    ↪ SocialLogin
# Facepy a plugin for Facebook
from facepy import GraphAPI
import requests
# REDFLIB
import rdflib
from rdflib import Namespace, Literal, RDFS, URIRef, BNode
from rdflib.namespace import RDF, FOAF
# get setting detail
from socialawareness.settings import BASE_DIR
import logging

logger = logging.getLogger(__name__)

# load ontology
```

```
logger.info("Loading␣Ontology")
ONTO=rdflib.Graph()
ONTO.load(BASE_DIR+'/socialContext.owl')
#PREFIX SC = Social Context
SC = Namespace("http://www.semanticweb.org/xgc4811/ontologies/2016/9/
    ↪ socialContext#")

# Persing the ontology data
parsing_to_str = lambda args: str(args).replace("['",'').replace("']",
    ↪ '')
remove_space = lambda args: str(args).replace("␣", "_").lower()

class FacebookManager (object):

def __init__(self, user=None):
logger.info("Facebook␣Manager␣Started")
self.user = None
self.social_access_token = None
if user is not None:
self.user = user
try:
self.social_access_token = SocialToken.objects.get(account__user=self.
    ↪ user, account__provider='facebook')
except ObjectDoesNotExist:
logger.error("Authentication␣with␣social␣account␣required")

def __repr__(self):
return repr(self.social_access_token)

def get_user_token (self):
return self.social_access_token

def get_user_detail(self):
if self.social_access_token != None:
graph = GraphAPI(self.social_access_token)
# Get Frineds
me= graph.get('me?fields=id,name,gender,picture,email,hometown,
    ↪ location,birthday')
return me

def get_user_friends(self):
allfriends =[]
if self.social_access_token != None:
graph = GraphAPI(self.social_access_token)
# Get Frineds taggable friends
# friends= graph.get('me/invitable_friends')
friends= graph.get('me/taggable_friends')
# Wrap this block in a while loop so we can keep paginating requests
    ↪ until
# finished.
while(True):
try:
for friend in iter(friends['data']):
allfriends.append(dict(name=friend['name'],
id=friend['id'],
url=friend['picture']['data']['url']))
# Attempt to make a request to the next page of data, if it exists.
```

```
friends=requests.get(friends['paging']['next']).json()
except KeyError:
# When there are no more pages (['paging']['next']), break from the
# loop and end the script.
break
return allfriends

class OntologyManager (object):

def __init__(self, user=None):
#logger.info("Ontology Manager Started")
global ONTO
global SC
self.user = None
if user is not None:
try:
email = URIRef("http://www.semanticweb.org/xgc4811/ontologies/2016/9/
    ↪ socialContext#%s"%str(user).lower())
qres = ONTO.query(
"""SELECT ?person
WHERE {
?person foaf:mbox ?email.
}""",initNs = { "foaf": FOAF}, initBindings = {"email": email})

for row in qres:
self.user = URIRef("%s"%row)
except ValueError:
print("The␣user␣does␣not␣exist␣in␣the␣ontology␣because␣authentication␣
    ↪ with␣social␣account␣required")

def add_user (self,**kwargs):
self.user = URIRef("http://www.semanticweb.org/xgc4811/ontologies
    ↪ /2016/9/socialContext#%s"%str(kwargs['email']).lower())
ONTO.add ((self.user, RDF.type, FOAF.Person))
ONTO.add ((self.user, FOAF.name, Literal(kwargs['name'])))
ONTO.add ((self.user, FOAF.firstName, Literal(kwargs['first_name'])))
ONTO.add ((self.user, FOAF.lastName, Literal(kwargs['last_name'])))
ONTO.add ((self.user, FOAF.gender, Literal(kwargs['gender'])))
ONTO.add ((self.user, FOAF.mbox, self.user))
ONTO.add ((self.user, SC.facebookID, Literal(kwargs['id'])))
ONTO.add ((self.user, SC.imageURL, Literal(kwargs['url'])))
ONTO.serialize("socialContext.owl", format="pretty-xml")
logger.info("User␣is␣added␣successfully␣")

def get_name (self, user):
qres = ONTO.query(
"""SELECT ?name
WHERE {
?person foaf:name ?name.
}""",initNs = { "foaf": FOAF}, initBindings = {"person": user})

for row in qres:
name = str ("%s"%row)
return name

def add_friend (self,**kwargs):
if self.user is not None:
```

```
friend = URIRef("http://www.semanticweb.org/xgc4811/ontologies/2016/9/
    ↪ socialContext#%s"%remove_space(kwargs['name']))
ONTO.add ((friend, RDF.type, SC['FriendList']))
ONTO.add ((friend, FOAF.name, Literal(kwargs['name'])))
ONTO.add ((friend, SC.facebookID, Literal(kwargs['id'])))
ONTO.add ((friend, SC.imageURL, Literal(kwargs['url'])))
ONTO.add ((friend, SC.isFriendOf, self.user))
ONTO.add ((self.user, SC.isFriendOf, friend))
ONTO.serialize("socialContext.owl", format="pretty-xml")
logger.info("Friend␣is␣added␣successfully␣")
else:
raise ValueError ("Cannot␣add␣friends␣because␣user␣is␣not␣found")

def get_friends (self):
onto_friendlist = []
if self.user is not None:
# Get friends using SPARQL
qres = ONTO.query(
"""SELECT ?name
WHERE {
?person1 sc:isFriendOf ?friend.
?friend foaf:name ?name.
}""",initNs = { "foaf": FOAF , "sc": "http://www.semanticweb.org/
    ↪ xgc4811/ontologies/2016/9/socialContext#"}, initBindings = {"
    ↪ person1": self.user})
for row in qres:
onto_friendlist.append("%s"%row)
return onto_friendlist
else:
raise ValueError ("Cannot␣get␣friends␣because␣user␣is␣not␣found")

def add_bluetooth(self, bluetooth):
if self.user is not None:
for b in ONTO.objects (self.user, FOAF.name):
ONTO.set((self.user,SC.bluetoothID, Literal(bluetooth)))
ONTO.serialize("socialContext.owl", format="pretty-xml")
logger.info("Bluetooth␣is␣added␣successfully␣")
else:
raise ValueError ("Cannot␣create␣bluetooth␣because␣user␣is␣not␣found")

def get_user_by_bluetooth (self, bluetooth):
# Get user by bluetooth ID using SPARQL
bluetooth = Literal(bluetooth)
user = None
qres = ONTO.query(
"""SELECT ?person
WHERE {
?person sc:bluetoothID ?bluetooth.
}""",initNs = {"sc": "http://www.semanticweb.org/xgc4811/ontologies
    ↪ /2016/9/socialContext#"}, initBindings = {"bluetooth": bluetooth
    ↪ })

for row in qres:
user = URIRef("%s"%row)
return user

# Getting common firends using python code
```

```
def get_common_friends (self, user_uri = None, bluetooth = None):
onto_friendlist1 = self.get_friends()
onto_friendlist2 = []
if bluetooth is not None:
user_uri = self.get_user_by_bluetooth(bluetooth)

if user_uri is not None:
for onto_friend in ONTO.objects (user_uri, SC.isFriendOf):
for onto_friend_name in ONTO.objects (onto_friend, FOAF.name):
onto_friendlist2.append("%s"%onto_friend_name)
common_friends = set(onto_friendlist1) & set(onto_friendlist2)
return common_friends
else:
print ("Could not find such a user in the dataset")

# Getting common friends using SPARQL
def get_common_friends_sec (self, user_uri= None, bluetooth = None):
common_friends = []
if bluetooth is not None:
user_uri = self.get_user_by_bluetooth(bluetooth)
if (user_uri is not None and self.user is not None):
qres = ONTO.query(
"""SELECT ?name
WHERE {
?person1 sc:isFriendOf ?friend.
?person2 sc:isFriendOf ?friend.
?friend foaf:name ?name
}""",initNs = { "foaf": FOAF , "sc": "http://www.semanticweb.org/
    ↪ xgc4811/ontologies/2016/9/socialContext#"}, initBindings = {"
    ↪ person1": self.user, "person2":user_uri})

for row in qres:
common_friends.append("%s"%row)
return common_friends
else:
print ("Could not find such a user in the dataset")

# Check if the users are already friends
def check_already_friends (self, user_uri= None, bluetooth = None ):
print (user_uri)
if bluetooth is not None:
user_uri = self.get_user_by_bluetooth(bluetooth)

if (user_uri is not None and self.user is not None):
qres = ONTO.query(
"""SELECT ?name
WHERE {
?person2 foaf:name ?name.
?person1 sc:isFriendOf ?friend.
?friend foaf:name ?name.
}
""",initNs = { "foaf": FOAF , "sc": "http://www.semanticweb.org/
    ↪ xgc4811/ontologies/2016/9/socialContext#"}, initBindings = {"
    ↪ person1": self.user, "person2":user_uri})

for row in qres:
return "%s"%row
```

```
else:
print ("Could not find such user in the dataset")
```

15. Now, Signup, and Login need to be managed in `model.py` file, this allows to acquire social contexts from Facebook, and store them into the ontology before Siginup, and Login are completed.

Listing A.5: commonfriends/model.py

```
from __future__ import unicode_literals

from django.db import models
# Create your models here.
from django.dispatch import receiver
# allauth
from allauth.account.signals import user_logged_in, user_signed_up
from .helper import ONTO, OntologyManager , FacebookManager
import logging

logger = logging.getLogger(__name__)

# Listen to new user and add them to ontology
@receiver(user_signed_up)
def user_signed_up(request, user, sociallogin=False, **kwargs):
if sociallogin:
logger.info("Getting user detail from Facebook for signing up")
# get user detail from facebook
fb = FacebookManager(user)
fb_user = fb.get_user_detail()
logger.info("Signing up new user in ontology")
# Create a new user in ontology user
# OntologyManager class should have no parameter
onto = OntologyManager()
onto.add_user(email= fb_user['email'],name=fb_user['name'], first_name
    ↪ = user.first_name, last_name= user.last_name,
id=fb_user['id'],
gender=fb_user['gender'],
url=fb_user['picture']['data']['url'])

# Listen to loggin and check if facebook friend has been changed
@receiver(user_logged_in)
def user_logged_in(request, user, sociallogin=None, **kwargs):
if sociallogin:

logger.info("Getting the current user detail from facebook")
fb = FacebookManager(user)
fb_friendlist = fb.get_user_friends()

logger.info("Getting the current user detail from ontology")
onto = OntologyManager(user.email)
onto_friendlist= onto.get_friends()

logger.info("Comparing between ontology friends and facebook friends")
```

```
for friend in fb_friendlist:
if not friend['name'] in onto_friendlist:
logger.info("Adding new a friend")
onto.add_friend(name=friend['name'],id=friend['id'], url=friend['url'
    ↪ ])
```

16. APIs need to be created in order to receive user's Bluetooth MAC address, and to search the ontology for MAC address, when the mobile application send them in `view.py` file.

Listing A.6: commonfriends/view.py

```
from django.shortcuts import render

# Create your views here.
from rest_framework.response import Response
from rest_framework import status
from rest_framework.views import APIView
from rest_framework.permissions import IsAuthenticated, AllowAny
from rest_framework.parsers import JSONParser
from django.core.exceptions import ObjectDoesNotExist
import logging
from .helper import OntologyManager, parsing_to_str
import json

logger = logging.getLogger(__name__)

#format of json;
# curl -X POST  -H "Content-Type: application/json" -H 'Authorization:
    ↪  Token d930a6d3e0fa7698eeeb956c0ea0ac047bc76d2d' -d "{'bluetooth
    ↪ ':''}" http://localhost:8000/api/bluetooth/user/
# Path http://localhost:8000/api/bluetooth/user/
class UserBluetooth(APIView):
permission_classes = (IsAuthenticated,)

def dispatch(self, *args, **kwargs):
return super(UserBluetooth, self).dispatch(*args, **kwargs)

def post(self, request):
logger.info("Register User Bluetooth")
data = request.data
user_bluetooth = data.get('user_bluetooth','')
# Checking if user_bluetooth is submitted
if not user_bluetooth:
logger.info("User Bluetooth is empty")
return Response(status = status.HTTP_400_BAD_REQUEST, data={'status':
    ↪ status.HTTP_400_BAD_REQUEST})
original_request = request._request
user = original_request.user
onto = OntologyManager(user.email)
if onto.user is None:
data={
'status': '400',
```

```
'data': 'Authentication with social account required '
}
return Response(status = status.HTTP_400_BAD_REQUEST, data=data)
onto.add_bluetooth(user_bluetooth)

return Response(status = status.HTTP_200_OK,data={'status':status.
    ↪ HTTP_200_OK })

#format of json:
# curl -X POST  -H "Content-Type: application/json" -H 'Authorization:
    ↪  Token d930a6d3e0fa7698eeeb956c0ea0ac047bc76d2d' -d "{'bluetooth
    ↪ ':''}" http://localhost:8000/api/bluetooth/search/

# Path http://localhost:8000/api/bluetooth/search/
class SearchFriendByBluetooth(APIView):
permission_classes = (IsAuthenticated,)

def dispatch(self, *args, **kwargs):
return super(SearchFriendByBluetooth, self).dispatch(*args, **kwargs)
def post(self, request):
# Friend status
NO_MATCH_FOUND = 0
USER_ALREADY_FRIEND =1
FOUND_MUTUAL_FRIENDS = 2
SUGGEST_FRIEND =3

logger.info("Receiving Surrounding Bluetooth")
data = request.data
bluetooth = data.get('bluetooth','')
# Checking if bluetooth is submitted
if not bluetooth:
logger.info("Bluetooth is empty")
return Response(status = status.HTTP_400_BAD_REQUEST, data={'status':
    ↪ status.HTTP_400_BAD_REQUEST})

# Getting the current user detail from ontology
original_request = request._request
user = original_request.user
onto_user = OntologyManager(user.email)

# Getting user info by bluetooth mac address
onto_user2 = OntologyManager()
bluetooth_user2 = onto_user2.get_user_by_bluetooth (bluetooth)

if bluetooth_user2:
# Checking if they are already friends
already_friends = onto_user.check_already_friends (None, bluetooth)

if already_friends:
data = {
'status':'200',
'user': already_friends,
'friend_status': USER_ALREADY_FRIEND,
'friend': [],
}
return Response(status = status.HTTP_200_OK,data=data)
```

```
# Checking if No mutual friend found
mutual = onto_user.get_common_friends_sec(None, bluetooth )

if mutual:
data = {
'status':'200',
'user': onto_user2.get_name(bluetooth_user2),
'friend_status': FOUND_MUTUAL_FRIENDS,
'friend': mutual,
}
return Response(status = status.HTTP_200_OK,data=data)

# if they are not frined and they do not have mutual friends
# Then suggest a friend
data = {
'status':'200',
'user': onto_user2.get_name(bluetooth_user2),
'friend_status': SUGGEST_FRIEND,
'friend': [],
}
return Response(status = status.HTTP_200_OK,data=data)
else:
# Otherwise return nothing
data = {
'status':'204',
'friend_status': NO_MATCH_FOUND,
'friend': [],
}
return Response(status = status.HTTP_204_NO_CONTENT, data= data)
```

17. APIs need to be created to receive the user's token and to permit the framework to recognize when they login with Facebook account and when they logout from mobile application. Navigate to `view.py` file under `facebookapi`.

Listing A.7: view.py

```
from django.shortcuts import render

# Create your views here.
from rest_framework.response import Response
from rest_framework import status
from rest_framework.views import APIView
from rest_framework.permissions import IsAuthenticated, AllowAny,
    ↪ IsAuthenticatedOrReadOnly
from rest_framework.parsers import JSONParser
from rest_framework.authtoken.models import Token
from django.contrib.auth import logout as auth_logout
# Allauth
from allauth.socialaccount import  providers
from allauth.socialaccount.models import SocialToken, SocialApp,
    ↪ SocialLogin
from allauth.socialaccount.providers.facebook.views import
    ↪ fb_complete_login
```

```
from allauth.socialaccount.helpers import complete_social_login
# Custome method
from .serializers import EverybodyCanAuthentication
import logging
from commonfriends.helper import FacebookManager

logger = logging.getLogger(__name__)

# Path= /api/login/facebook

class RestFacebookLogin (APIView):
permission_classes = (AllowAny,)
authentication_classes = (EverybodyCanAuthentication,)

def dispatch(self, *args, **kwargs):
        return super(RestFacebookLogin, self).dispatch(*args, **kwargs
            ↪ )

def post (self,request):

original_request = request._request
data = JSONParser().parse(request)
access_token = data.get('access_token', '')

try:
app = SocialApp.objects.get(provider='facebook')
fb_auth_token = SocialToken(app=app, token=access_token)

login = fb_complete_login(original_request, app, fb_auth_token)
login.token = fb_auth_token
login.state = SocialLogin.state_from_request(original_request)

complete_social_login(original_request, login)
token, _ = Token.objects.get_or_create(user=original_request.user)

data_response ={
'username': original_request.user.username,
'objectId': original_request.user.pk,
'firstName': original_request.user.first_name,
'lastName': original_request.user.last_name,
'email': original_request.user.email,
'sessionToken': token.key,
}
return Response(status=status.HTTP_200_OK, data=data_response)
except:
return Response(status=status.HTTP_401_UNAUTHORIZED,data={
'detail': 'Bad Access Token',
})

# Path = /api/logout/facebook/

class RestFacebookLogout (APIView):

def dispatch(self, *args, **kwargs):
return super(RestFacebookLogout, self).dispatch(*args, **kwargs)

def post(self, request):
```

```
original_request = request._request
if original_request.user.is_authenticated():
auth_logout(original_request)
request.user.auth_token.delete()
return Response (status=status.HTTP_200_OK, data= {'status': status.
    ↪ HTTP_200_OK })
```

18. Finally, add URL for allauth and the APIs that we created in the above steps in `url.py` and so it looks as the following:

```
from django.conf.urls import url, include
from django.contrib import admin
from rest_framework import routers
from django.views.decorators.csrf import csrf_exempt
from commonfriends.views import UserBluetooth, SearchFriendByBluetooth
    ↪ , OwlReadyOntology
from facebookapi.views import FacePy, RestFacebookLogin,
    ↪ RestFacebookLogout




urlpatterns = [
        url(r'^admin/', admin.site.urls),
        url(r'^accounts/', include('allauth.urls')),
        url(r'^api-auth/', include('rest_framework.urls', namespace='
            ↪ rest_framework')),
        url(r'^api/bluetooth/user/$', csrf_exempt(UserBluetooth.
            ↪ as_view())),
        url(r'^api/bluetooth/search/$', csrf_exempt(
            ↪ SearchFriendByBluetooth.as_view())),
        url(r'^api/ontology/', OwlReadyOntology.as_view()),
        url(r'^api/facepy/', FacePy.as_view()),
        url(r'^api/login/facebook/$', csrf_exempt(RestFacebookLogin.
            ↪ as_view()), name='rest-facebook-login'),
        url(r'^api/logout/facebook/$', csrf_exempt(RestFacebookLogout.
            ↪ as_view()), name='rest-facebook-logout'),
]
```

A.3 Deployment

1. Navigate to our project, and create `requirement.txt` file that contains a simple list of all the packages in the current environment, and their respective versions. To do that, write in the Terminal:

   `pip freeze > requirements.txt`

2. Navigate to `pythonanywhere.com` and create an account.
3. Upload our project; including `requirements.txt` that created in step 1. Then, use the simple user interface provided as shown in Figure A.3.

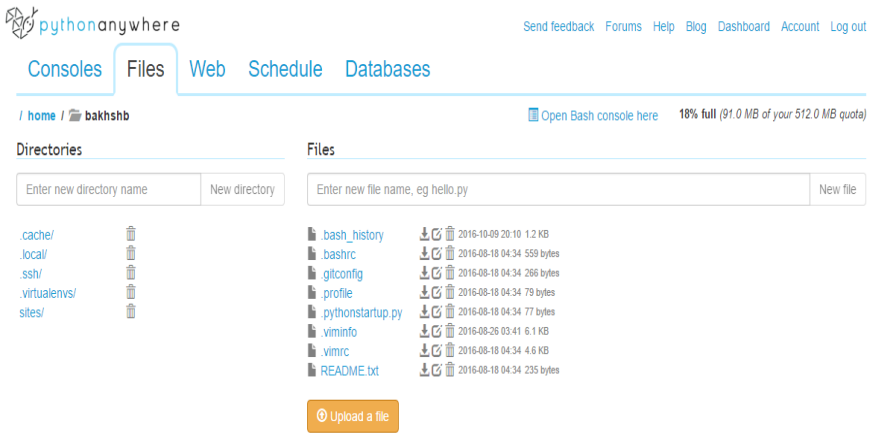


Figure A.3: Uploading Files in Pythonanywhere

4. Navigate to `Console` in Pythonanywhere, and click on `Bash` to open a Terminal.
5. Create a virtual environment as demonstrated in Section A.1.
6. Install the packages from `requirements.txt`into the virtual environment:

```
pip install -r requirements.txt
```

7. Navigate to `Web` in Pythonanywhere, and under code, click on `Source Code` to set the path of our project as shown in Figure A.4.

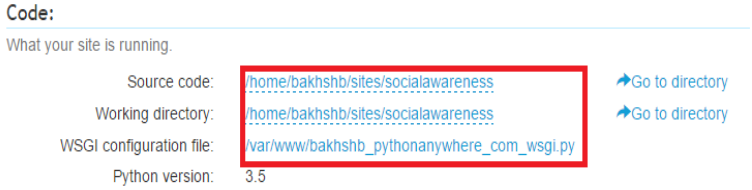


Figure A.4: WSG in Pythonanywhere

8. Set the path for `Working directory`,which is the same as the project path.
9. Configure `WSG`; therefore, the server locates our project as follows.

```
import os
import sys

path = '/home/bakhshb/sites/socialawareness'  # use your own
    ↪ PythonAnywhere username here
if path not in sys.path:
sys.path.append(path)

os.environ['DJANGO_SETTINGS_MODULE'] = 'socialawareness.settings'

from django.core.wsgi import get_wsgi_application
#application = get_wsgi_application()
from django.contrib.staticfiles.handlers import StaticFilesHandler
application = StaticFilesHandler(get_wsgi_application())
```

Appendix B

Mobile Application Setup

B.1 Application Dependencies

1. Create a new App in Facebook Developer website.
2. Go to Android Studio | New Project | Minimum SDK.
3. Select API 15: Android 4.0.3 or higher and create your new project.
4. In our project, open our app | Gradle Scripts | build.gradle.
5. Add compile to build.gradle dependencies:

```
compile 'com.facebook.android:facebook-android-sdk:4.+'
```

```
compile 'com.android.volley:volley:1.0.0'
```

6. Build project.
7. Import the Facebook SDK, and Volly:

```
import com.facebook.FacebookSdk;
```

```
import com.android.volley.Request;
```

```
import com.android.volley.RequestQueue;
```

```
import com.android.volley.toolbox.Volley;
```

B.2 Project Setup

1. Open `strings.xml` file. For example:

   `/app/src/main/res/values/strings.xml`

2. Add a new string with the name `facebook_app_id` containing the value of our Facebook App ID:

   `<string name="facebook_app_id">1012388762163162</string>`

3. Open `AndroidManifest.xml`.

4. Add a `user-permission` element to the manifest:

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
<uses-permission android:name="android.permission.
    ↪ ACCESS_COARSE_LOCATION"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE
    ↪ "/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
```

5. Add meta-data elements to the application element:

Listing B.1: Application Elements

```
<application
android:name=".MyApplication"
android:allowBackup="true"
android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:supportsRtl="true"
android:theme="@style/AppTheme">
<meta-data
android:name="com.facebook.sdk.ApplicationId"
android:value="@string/facebook_app_id"/>
</application>
```

6. Add Activities to your AndroidManifest.xml:

Listing B.2: AndroidManifest

```
<activity android:name=".activities.LoginActivity">
<intent-filter>
<action android:name="android.intent.action.MAIN"/>
<category android:name="android.intent.category.LAUNCHER"/>
</intent-filter>
</activity>
<activity android:name=".activities.BluetoothActivity">
</activity>
<receiver
android:name=".receivers.NetworksReceiver"
android:label="BluetoothReceiver">
<intent-filter>
<action android:name="android.bluetooth.adapter.action.STATE_CHANGED
    ↪ "/>
<action android:name="android.bluetooth.adapter.action.
    ↪ CONNECTION_STATE_CHANGED"/>
<action android:name="android.bluetooth.device.action.ACL_CONNECTED"/>
<action android:name="android.bluetooth.device.action.ACL_DISCONNECTED
    ↪ "/>
<action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
</intent-filter>
</receiver>
<service android:name=".services.BluetoothService">
</service>
<activity android:name=".activities.ResultsActivity">
</activity>
<activity android:name=".activities.MainActivity">
</activity>
```

7. Add a new Activity `MainActivity`:

Listing B.3: MainActivity.java

```
package com.example.xgc4811.myapp.activities;

import android.Manifest;
import android.content.pm.PackageManager;
import android.support.annotation.NonNull;
import android.support.v4.app.ActivityCompat;
import android.support.v4.content.ContextCompat;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.Toast;
import com.example.xgc4811.myapp.R;
import com.example.xgc4811.myapp.fragments.MainFragment;

public class MainActivity extends AppCompatActivity {

// Request Location permission
private static final int PERMISSION_REQUEST_CODE = 1;
private static final String TAG = "MainActivity";

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
```

```
setContentView( R.layout.activity_main);
if(savedInstanceState == null){
//            LoginFragment fb_login = new LoginFragment();
MainFragment mainFragment = new MainFragment();
getSupportFragmentManager().beginTransaction().add(R.id.
    ↪ container_layout,mainFragment,"FB").commit();
}

// Check Location
if (!checkPermission()){
requestPermission();
}
}

private boolean checkPermission(){
int result = ContextCompat.checkSelfPermission(getApplicationContext()
    ↪ , Manifest.permission.ACCESS_COARSE_LOCATION);
if (result == PackageManager.PERMISSION_GRANTED){
return true;
} else {
return false;
}
}

private void requestPermission(){
if (ActivityCompat.shouldShowRequestPermissionRationale(this,Manifest.
    ↪ permission.ACCESS_COARSE_LOCATION)){
Toast.makeText(getApplicationContext(),"GPS␣permission␣allows␣us␣to␣
    ↪ access␣location␣data.␣Please␣allow␣in␣App␣Settings␣for␣
    ↪ additional␣functionality.", Toast.LENGTH_LONG).show();
} else {
ActivityCompat.requestPermissions(this,new String[]{Manifest.
    ↪ permission.ACCESS_COARSE_LOCATION},PERMISSION_REQUEST_CODE);
}
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull
    ↪ String[] permissions, @NonNull int[] grantResults) {
switch (requestCode) {
case PERMISSION_REQUEST_CODE:
if (grantResults.length > 0 && grantResults[0] == PackageManager.
    ↪ PERMISSION_GRANTED) {
Toast.makeText( getApplicationContext(), "Permission␣Granted,␣Now␣you␣
    ↪ can␣access␣location␣data", Toast.LENGTH_SHORT ).show();
} else {
Toast.makeText( getApplicationContext(), "Permission␣Denied,␣You␣
    ↪ cannot␣access␣location␣data", Toast.LENGTH_SHORT ).show();
}
break;
}
}
}
```

8. Add to `activity_main.xml` layout:

Listing B.4: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
    ↪ android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context=".activities.MainActivity">


<FrameLayout
android:id="@+id/container_layout"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:layout_alignParentTop="true"
android:layout_alignParentStart="true"></FrameLayout>
</RelativeLayout>
```

9. Add a new Activity `LoginActivity`:

Listing B.5: LoginActivity.java

```
package com.example.xgc4811.myapp.activities;

import android.app.ProgressDialog;
import android.bluetooth.BluetoothAdapter;
import android.content.Intent;
import android.os.Handler;
import android.os.Message;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;
import com.android.volley.AuthFailureError;
import com.android.volley.DefaultRetryPolicy;
import com.android.volley.Request;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.JsonObjectRequest;
import com.example.xgc4811.myapp.R;
import com.example.xgc4811.myapp.helper.AppHelper;
import com.example.xgc4811.myapp.helper.CustomRequestQueue;
import com.facebook.AccessToken;
import com.facebook.CallbackManager;
import com.facebook.FacebookCallback;
import com.facebook.FacebookException;
import com.facebook.FacebookSdk;
import com.facebook.login.LoginResult;
import com.facebook.login.widget.LoginButton;
```

```
import org.json.JSONException;
import org.json.JSONObject;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

public class LoginActivity extends AppCompatActivity {

private static final String TAG = "LoginActivity";

//System status
static final int STATE_NONE=0;
static final int STATE_LOGGING_IN = 1;

// API
private static final String LOGIN_URL = "http://bakhshb.pythonanywhere
    ↪ .com/api/login/facebook/";

private AppHelper mAppHelper;
private TextView mTextView;
private ProgressDialog mProgressDialog;

//Facebook
private CallbackManager callbackManager;
private FacebookCallback<LoginResult> facebookCallBack = new
    ↪ FacebookCallback<LoginResult>() {
@Override
public void onSuccess(LoginResult loginResult) {
// Login to backend
AccessToken token = loginResult.getAccessToken();
if (token.getToken() != null) {
mHandler.obtainMessage( STATE_LOGGING_IN, token.getToken() ).
    ↪ sendToTarget();
}
}
@Override
public void onCancel() {

}
@Override
public void onError(FacebookException error) {
}
};

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate( savedInstanceState );
FacebookSdk.sdkInitialize(this.getApplicationContext());
callbackManager = CallbackManager.Factory.create();
setContentView( R.layout.activity_login );
this.setTitle( "Common␣Friends␣Login" );

init(  );

if (mAppHelper.isLoggedIn()){
Intent mIntentMainAcivity = new Intent( getApplicationContext(),
    ↪ MainActivity.class );
```

```
startActivity( mIntentMainAcivity );
finish();

}
}

public void init (){
LoginButton loginButton = (LoginButton) findViewById(R.id.
    ↪ facebook_login_button);
loginButton.setReadPermissions( Arrays.asList("public_profile","email"
    ↪ , "user_friends"));
loginButton.registerCallback(callbackManager, facebookCallBack);
mTextView = (TextView) findViewById(R.id.user_detail);
mTextView.setText( "Please␣Login␣With␣Facebook" );
mAppHelper = new AppHelper( this );
mProgressDialog = new ProgressDialog (this); //display an invisible
    ↪ overlay dialog to prevent user interaction and pressing back
mProgressDialog.setCancelable(false);
mProgressDialog.setProgressStyle( mProgressDialog.STYLE_SPINNER );
}

@Override
protected void onActivityResult(int requestCode, int resultCode,
    ↪ Intent data) {
super.onActivityResult( requestCode, resultCode, data );
callbackManager.onActivityResult(requestCode, resultCode, data);
}

private final Handler mHandler = new Handler(  ){
@Override
public void handleMessage(Message msg) {
switch (msg.what){
case STATE_NONE:
Log.i( TAG, "handleMessage:␣Clear" );
break;
case STATE_LOGGING_IN:
if (turnBluetooth( true ) == true) {
loginRequest( (String) msg.obj );
}
Log.i( TAG, "handleMessage:␣Login␣in␣to␣server" );
break;
}
}
};

private boolean turnBluetooth (boolean bluetooth){
BluetoothAdapter mBluetoothAdapter;
if (bluetooth == true) {
// turn bluetooth off
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter.isEnabled()) {
}else{
mBluetoothAdapter.enable();
}
return true;
} else {
// turn bluetooth off
```

```
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter.isEnabled()) {
mBluetoothAdapter.disable();
}
return false;
}
}

private void loginRequest (String facebookToken){
mProgressDialog.setMessage("Please␣wait␣while␣Logging␣in␣...");
showDialog();
HashMap<String, String> params = new HashMap<String, String>();
params.put("access_token", facebookToken);
JsonObjectRequest jsonObjectRequest = new JsonObjectRequest( Request.
    ↪ Method.POST, LOGIN_URL, new JSONObject( params ), new Response.
    ↪ Listener<JSONObject>() {
@Override
public void onResponse(JSONObject response) {
hideDialog();
try {
mAppHelper.createLoginSession( response.getString( "sessionToken" ) );
//save bluetooth mac address
mAppHelper.setDeviceAddress( android.provider.Settings.Secure.
    ↪ getString(getContentResolver(), "bluetooth_address") );
Intent mIntentMainAcivity = new Intent( getApplicationContext(),
    ↪ MainActivity.class );
startActivity( mIntentMainAcivity );
finish();
} catch (JSONException e) {
e.printStackTrace();
}
}
}, new Response.ErrorListener() {
@Override
public void onErrorResponse(VolleyError error) {
hideDialog();
}
} ){
@Override
public Map<String, String> getHeaders() throws AuthFailureError {
HashMap<String, String> headers = new HashMap<String, String>();
headers.put("Content-Type","application/json");
return headers;
}
};

jsonObjectRequest.setRetryPolicy( new DefaultRetryPolicy( 3000,
    ↪ DefaultRetryPolicy.DEFAULT_MAX_RETRIES, DefaultRetryPolicy.
    ↪ DEFAULT_BACKOFF_MULT ) );
CustomRequestQueue.getInstance( this ).addToRequestQueue(
    ↪ jsonObjectRequest );
}

private void showDialog() {
if (!mProgressDialog.isShowing())
mProgressDialog.show();
}
```

```
private void hideDialog() {
if (mProgressDialog.isShowing())
mProgressDialog.dismiss();
}
}
```

10. Add to `activity_main.xml` layout:

Listing B.6: activity_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
    ↪ android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".activities.LoginActivity">
<TextView
android:id="@+id/user_detail"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_gravity="center_horizontal|top"
android:layout_marginTop="80dp"
android:textSize="30dp"
android:textAlignment="center"/>
<RelativeLayout
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_centerVertical="true"
android:layout_centerHorizontal="true"
android:id="@+id/relativeLayout">
<com.facebook.login.widget.LoginButton
android:id="@+id/facebook_login_button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center_horizontal|center"/>
</RelativeLayout>
</RelativeLayout>
```

11. Add a new Activity `BluetoothActivity`:

Listing B.7: BluetoothActivity.java

```
package com.example.xgc4811.myapp.activities;

import android.app.ProgressDialog;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
```

```
import android.content.IntentFilter;
import android.os.Handler;
import android.os.Message;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
import com.example.xgc4811.myapp.services.BluetoothChatService;
import com.example.xgc4811.myapp.Constants;
import com.example.xgc4811.myapp.R;
import java.util.ArrayList;
import java.util.Set;
import java.util.UUID;

public class BluetoothActivity extends AppCompatActivity implements
    ↪ AdapterView.OnItemClickListener {

// Debugging
private static final String TAG = "BluetoothChat";
private static final boolean D = true;
// Intent request codes
private static final int REQUEST_CONNECT_DEVICE = 1;
private static final int REQUEST_ENABLE_BT = 2;

// Arrays
ArrayAdapter<String> listAdapter;
ArrayList<String> pairedDevices;
ArrayList<BluetoothDevice> devices;
// UI
TextView mTextView;
ListView mListView ;
ProgressDialog mProgressDialog;
Button mButton;


// Bluethooth
BluetoothAdapter mBluetoothAdapter;
Set<BluetoothDevice> devicesArray;
IntentFilter mIntentFilter;
BroadcastReceiver mBroadcastReceiver;
public static final UUID MY_UUID = UUID.fromString( "8ce255c0-200a-11
    ↪ e0-ac64-0800200c9a66" );


BluetoothChatService mBluetoothChatService ;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate( savedInstanceState );
setContentView( R.layout.activity_bluetooth );
this.setTitle( "Search␣Bluetooth" );
```

```
init();

if (mBluetoothAdapter == null){
Toast.makeText(getApplicationContext(), "No␣Bluethooth␣detected", 0).
    ↪ show();
finish();
}else{
if (!mBluetoothAdapter.isEnabled()){
turnBluetoothOn();
}

getPairedDevices();
startDiscovery();
}
}

private void startDiscovery() {
if (mBluetoothAdapter.isDiscovering()) {
mBluetoothAdapter.cancelDiscovery();
}
mBluetoothAdapter.startDiscovery();
}

private void turnBluetoothOn() {
Intent intent = new Intent( BluetoothAdapter.ACTION_REQUEST_ENABLE );
startActivityForResult( intent,1 );
}

private void getPairedDevices() {
devicesArray = mBluetoothAdapter.getBondedDevices();
if (devicesArray.size() >0){
for (BluetoothDevice device : devicesArray){
pairedDevices.add( device.getName());
}
}
}


private void init(){
mProgressDialog = new ProgressDialog( this );
mProgressDialog.setMessage( "Searching␣for␣devices" );
mProgressDialog.setProgressStyle( mProgressDialog.STYLE_SPINNER );
mProgressDialog.setCancelable( false );

mTextView = (TextView) findViewById(R.id.response_message);
mListView = (ListView) findViewById( R.id.listView );
mListView.setOnItemClickListener( this );
mButton = (Button) findViewById( R.id.button );

pairedDevices = new ArrayList<String>( );
devices = new ArrayList<BluetoothDevice>(  );
listAdapter = new ArrayAdapter<String>( this, android.R.layout.
    ↪ simple_list_item_1,0 );
mListView.setAdapter( listAdapter );
```

```
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
mIntentFilter = new IntentFilter( BluetoothDevice.ACTION_FOUND );

mBroadcastReceiver = new BroadcastReceiver() {
@Override
public void onReceive(Context context, Intent intent) {
String action = intent.getAction();
if (BluetoothDevice.ACTION_FOUND.equals( action )){
BluetoothDevice device = intent.getParcelableExtra( BluetoothDevice.
    ↪ EXTRA_DEVICE );
devices.add( device );
listAdapter.add( device.getName()+"" + "\n"+device.getAddress());
}
else if (BluetoothAdapter.ACTION_DISCOVERY_STARTED .equals( action )){
mProgressDialog.show();
devices.clear();
listAdapter.clear();
}
else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED .equals( action ))
    ↪ {
mProgressDialog.dismiss();
mTextView.setText( "Available␣Devices" );
}
else if (BluetoothAdapter.ACTION_STATE_CHANGED .equals( action )){
if (mBluetoothAdapter.getState() == mBluetoothAdapter.STATE_OFF){
turnBluetoothOn();
}
}
}
};
registerReceiver(mBroadcastReceiver,mIntentFilter );

mIntentFilter =  new IntentFilter( BluetoothAdapter.
    ↪ ACTION_DISCOVERY_STARTED );
registerReceiver(mBroadcastReceiver,mIntentFilter );
mIntentFilter =  new IntentFilter( BluetoothAdapter.
    ↪ ACTION_DISCOVERY_FINISHED );
registerReceiver(mBroadcastReceiver,mIntentFilter );
mIntentFilter =  new IntentFilter( BluetoothAdapter.
    ↪ ACTION_STATE_CHANGED );
registerReceiver(mBroadcastReceiver,mIntentFilter );

mBluetoothChatService = new BluetoothChatService( this, mHandler );

mButton.setOnClickListener( new View.OnClickListener() {
@Override
public void onClick(View v) {
String message = "Hello";
byte[] send = message.getBytes();
mBluetoothChatService.write( send );
}
} );
}


@Override
```

```
protected void onActivityResult(int requestCode, int resultCode,
    ↪ Intent data) {
super.onActivityResult( requestCode, resultCode, data );
if (resultCode == RESULT_CANCELED){
Toast.makeText(getApplicationContext(), "Bluethooth must be enabled to
    ↪ carry on", Toast.LENGTH_SHORT ).show();
finish();
}else{
Intent discoverableIntent = new
Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
discoverableIntent.putExtra(BluetoothAdapter.
    ↪ EXTRA_DISCOVERABLE_DURATION, 300);
startActivity(discoverableIntent);
startDiscovery();
mBluetoothChatService.start();
}
}

@Override
protected void onResume() {
super.onResume();
IntentFilter quitFilter = new IntentFilter();
quitFilter.addAction(BluetoothDevice.ACTION_FOUND);
registerReceiver(mBroadcastReceiver, quitFilter);
}

@Override
protected void onDestroy() {
super.onDestroy();
mBluetoothAdapter = null;
unregisterReceiver( mBroadcastReceiver );
if (mBluetoothChatService != null){
mBluetoothChatService.stop();
}
}


@Override
public void onItemClick(AdapterView<?> parent, View view, int position
    ↪ , long id) {
if (mBluetoothAdapter.isDiscovering()){
mBluetoothAdapter.cancelDiscovery();
}
BluetoothDevice selectedDevice = devices.get( position );
if (mBluetoothChatService != null){
mBluetoothChatService.stop();
}
mBluetoothChatService.start();
mBluetoothChatService.connect( selectedDevice );
}
private Handler mHandler = new Handler(  ){
@Override
public void handleMessage(Message msg) {
switch(msg.what){
case Constants.MESSAGE_WRITE:
byte[] writeBuf = (byte[]) msg.obj;
// construct a string from the buffer
```

```
String writeMessage = new String(writeBuf);
Log.d( TAG, "handleMessage:␣" + writeMessage );
Toast.makeText(getApplicationContext(), writeMessage,
Toast.LENGTH_SHORT).show();
break;
case Constants.MESSAGE_READ:
byte[] readBuf = (byte[]) msg.obj;
// construct a string from the valid bytes in the buffer
String readMessage = new String(readBuf, 0, msg.arg1);
Toast.makeText(getApplicationContext(), readMessage,
Toast.LENGTH_SHORT).show();
Log.d( TAG, "handleMessage:␣" + readMessage );
break;
case Constants.MESSAGE_TOAST:
Toast.makeText(getApplicationContext(), msg.getData().getString(
    ↪ Constants.TOAST),
Toast.LENGTH_SHORT).show();
break;
}
}
};
}
```

12. Add to `activity_bluetooth.xml` layout:

Listing B.8: activity_bluetooth.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
android:paddingTop="@dimen/activity_vertical_margin"
tools:context=".activities.BluetoothActivity">

<TextView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Bluetooth"
android:id="@+id/response_message"
android:layout_alignParentTop="true"/>

<ListView
android:layout_below="@+id/response_message"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@+id/listView"/>

<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
```

```
android:text="New Button"
android:id="@+id/button"
android:layout_alignParentBottom="true"
android:layout_alignParentEnd="true"/>

</RelativeLayout>
```

13. Add a new Activity `ResultActivity`:

Listing B.9: ResultActivity.java

```
package com.example.xgc4811.myapp.activities;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;
import com.example.xgc4811.myapp.R;

public class ResultsActivity extends AppCompatActivity {

private static final String TAG = "ResultsAcitivty";
private TextView mTextView;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate( savedInstanceState );
setContentView( R.layout.activity_bluetooth_results );
this.setTitle( "Bluetooth Result" );
Bundle mBundle = getIntent().getExtras();
mTextView = (TextView) findViewById( R.id.bluetoothResults );
mTextView.setText( "" );
if (mBundle != null){
Log.d( TAG, "" + mBundle.get( "msg" )  );
mTextView.setText( mBundle.getString( "msg" ) );
}
}
}
```

14. Add to `activity_bluetooth_results.xml` layout:

Listing B.10: activity_bluetooth_results.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="@dimen/activity_vertical_margin"
android:paddingLeft="@dimen/activity_horizontal_margin"
android:paddingRight="@dimen/activity_horizontal_margin"
```

```
android:paddingTop="@dimen/activity_vertical_margin"
tools:context=".activities.ResultsActivity">
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textAppearance="?android:attr/textAppearanceLarge"
android:text="Large Text"
android:id="@+id/bluetoothResults"
android:layout_centerVertical="true"
android:layout_centerHorizontal="true"
android:textAlignment="center"/>
</RelativeLayout>
```

15. Add a new Fragment `MainFragment`:

Listing B.11: MainFragment

```
package com.example.xgc4811.myapp.fragments;

import android.app.AlarmManager;
import android.app.PendingIntent;
import android.app.ProgressDialog;
import android.bluetooth.BluetoothAdapter;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Button;
import android.widget.TextView;

import com.android.volley.AuthFailureError;
import com.android.volley.DefaultRetryPolicy;
import com.android.volley.Request;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.JsonObjectRequest;
import com.example.xgc4811.myapp.R;
import com.example.xgc4811.myapp.activities.LoginActivity;
import com.example.xgc4811.myapp.activities.MainActivity;
import com.example.xgc4811.myapp.helper.AppHelper;
import com.example.xgc4811.myapp.helper.CustomRequestQueue;
import com.example.xgc4811.myapp.services.BluetoothService;
import com.facebook.Profile;
import com.facebook.login.LoginManager;

import org.json.JSONException;
import org.json.JSONObject;

import java.util.Calendar;
import java.util.HashMap;
import java.util.Map;
```

```
public class MainFragment extends Fragment implements View.
    ↪ OnClickListener {

private static final String TAG = "MainFragment";
// API
private static final String LOGOUT_URL = "http://bakhshb.
    ↪ pythonanywhere.com/api/logout/facebook/";
private static final String BLUETOOTH_USER_URL = "http://bakhshb.
    ↪ pythonanywhere.com/api/bluetooth/user/";
private AppHelper mAppHelper;
//Fragment
CommonFriendsFragment commonFriendsFragment = null;
UserAccountFragment userAccountFragment = null;

private TextView mTextView;
private Button mButtonTurnDiscovery;
private Button mButtonCommonFriends;
private Button mButtonAccount;
private Button mButtonLogout;

private ProgressDialog mProgressDialog;

public MainFragment() {
}

@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate( savedInstanceState );
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
return inflater.inflate( R.layout.fragment_main, container, false );
}

@Override
public void onViewCreated(View view, Bundle savedInstanceState) {
super.onViewCreated( view, savedInstanceState );
init( view );
if (!mAppHelper.isLoggedIn()){
bluetoothService( false );
turnBluetooth( false );
Intent mIntentMainAcivity = new Intent( this.getContext(),
    ↪ MainActivity.class );
startActivity( mIntentMainAcivity );
getActivity().finish();
}else{
BluetoothAdapter bAdapter = BluetoothAdapter.getDefaultAdapter();
if(bAdapter.getScanMode() == BluetoothAdapter.
    ↪ SCAN_MODE_CONNECTABLE_DISCOVERABLE) {
// device is discoverable & connectable
bluetoothService( true );
bluetoothUserRequest();
} else {
```

```
// device is not discoverable & connectable
bluetoothUserRequest();
// Making the device discoverable
Intent enableBtIntent = new Intent(BluetoothAdapter.
    ↪ ACTION_REQUEST_DISCOVERABLE);
enableBtIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION,
    ↪ 0);
startActivityForResult(enableBtIntent, 0);
bluetoothService( true );
}
}

}

public void init (View view){
mAppHelper = new AppHelper( getContext() );

commonFriendsFragment = new CommonFriendsFragment();

mTextView = (TextView) view.findViewById(R.id.user_detail);
mButtonTurnDiscovery = (Button) view.findViewById(R.id.
    ↪ button_turndiscovery);
mButtonTurnDiscovery.setOnClickListener( this );
mButtonCommonFriends = (Button) view.findViewById(R.id.
    ↪ button_commonfriends);
mButtonCommonFriends.setOnClickListener(this);
mButtonAccount = (Button) view.findViewById(R.id.button_account1);
mButtonAccount.setOnClickListener(this);
mButtonLogout = (Button) view.findViewById(R.id.button_logout);
mButtonLogout.setOnClickListener( this );

mProgressDialog = new ProgressDialog (getActivity()); //display an
    ↪ invisible overlay dialog to prevent user interaction and
    ↪ pressing back
mProgressDialog.setCancelable(false);
mProgressDialog.setProgressStyle( mProgressDialog.STYLE_SPINNER );
}

@Override
public void onClick(View v) {
switch (v.getId()){
case R.id.button_turndiscovery:
BluetoothAdapter bAdapter = BluetoothAdapter.getDefaultAdapter();
if(bAdapter.getScanMode() == BluetoothAdapter.
    ↪ SCAN_MODE_CONNECTABLE_DISCOVERABLE) {
// device is discoverable & connectable
turnBluetooth( false );
bluetoothService( false );
} else {
// device is not discoverable & connectable
// Making the device discoverable
Intent enableBtIntent = new Intent(BluetoothAdapter.
    ↪ ACTION_REQUEST_DISCOVERABLE);
enableBtIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION,
    ↪ 0);
startActivityForResult(enableBtIntent, 0);
bluetoothUserRequest();
```

```
bluetoothService( true );
}
break;
case R.id.button_commonfriends:
if (commonFriendsFragment != null){
getFragmentManager().beginTransaction().replace(R.id.container_layout,
    ↪ commonFriendsFragment).addToBackStack(null).commit();
}

break;
case R.id.button_account1:
if (userAccountFragment !=null) {
getFragmentManager().beginTransaction().replace(R.id.container_layout,
    ↪  userAccountFragment).addToBackStack(null).commit();
}
break;
case R.id.button_logout:
turnBluetooth( false );
bluetoothService( false );
LoginManager.getInstance().logOut();
logoutRequest();
break;
}
}

private void bluetoothService(boolean bluetoothStatus){
// Start bluetooth service using AlarmManager
Calendar cal = Calendar.getInstance();
cal.add( Calendar.SECOND, 10 );
Intent mintent = new Intent( getContext(), BluetoothService.class );
PendingIntent pintent = PendingIntent.getService( getContext(), 0,
    ↪ mintent, PendingIntent.FLAG_UPDATE_CURRENT );
AlarmManager alarm = (AlarmManager) getContext().getSystemService(
    ↪ Context.ALARM_SERVICE );
if (bluetoothStatus) {
//for 30 mint 60*60*1000
alarm.setRepeating( AlarmManager.RTC_WAKEUP, cal.getTimeInMillis(),
    ↪ 120 * 1000, pintent );
// bluetoothUserRequest();
}else if (!bluetoothStatus){
alarm.cancel( pintent );
getContext().stopService( new Intent( getContext(), BluetoothService.
    ↪ class ) );
}
}

private void bluetoothUserRequest(){
HashMap<String, String > user = mAppHelper.getUserDetails();
final String token = user.get( AppHelper.USER_TOKEN );
final String bluetooth_address = user.get( AppHelper.BLUETOOTH_ADDRESS
    ↪  );
Log.d( TAG, "bluetoothUserRequest: " + bluetooth_address );
//
HashMap<String, String> params = new HashMap<String, String>();
params.put("user_bluetooth", bluetooth_address);
```

```
JsonObjectRequest jsonObjectRequest = new JsonObjectRequest( Request.
    ↪ Method.POST, BLUETOOTH_USER_URL, new JSONObject( params ), new
    ↪ Response.Listener<JSONObject>() {
@Override
public void onResponse(JSONObject response) {
try {
Log.d( TAG, "onResponse: " + response.getString( "status" ) );
} catch (JSONException e) {
e.printStackTrace();
}
}
}, new Response.ErrorListener() {
@Override
public void onErrorResponse(VolleyError error) {
}
} ){
@Override
public Map<String, String> getHeaders() throws AuthFailureError {
HashMap<String, String> headers = new HashMap<String, String>();
headers.put("Authorization","Token "+ token);
headers.put("Content-Type","application/json");
return headers;
}
};
jsonObjectRequest.setRetryPolicy( new DefaultRetryPolicy( 3000,
    ↪ DefaultRetryPolicy.DEFAULT_MAX_RETRIES, DefaultRetryPolicy.
    ↪ DEFAULT_BACKOFF_MULT ) );
CustomRequestQueue.getInstance( getContext() ).addToRequestQueue(
    ↪ jsonObjectRequest);
}

private void logoutRequest(){
mProgressDialog.setMessage("Please wait while Logging out ...");
showDialog();
HashMap<String, String > user = mAppHelper.getUserDetails();
final String token = user.get( AppHelper.USER_TOKEN );
JsonObjectRequest jsonObjectRequest = new JsonObjectRequest( Request.
    ↪ Method.POST, LOGOUT_URL, null, new Response.Listener<JSONObject
    ↪ >() {
@Override
public void onResponse(JSONObject response) {
hideDialog();
mAppHelper.logoutUser();
Intent mIntentMainAcivity = new Intent( getContext(), LoginActivity.
    ↪ class );
startActivity( mIntentMainAcivity );
getActivity().finish();
}
}, new Response.ErrorListener() {
@Override
public void onErrorResponse(VolleyError error) {
hideDialog();
}
} ){
@Override
public Map<String, String> getHeaders() throws AuthFailureError {
HashMap<String, String> headers = new HashMap<String, String>();
```

```
headers.put("Authorization","Token "+ token);
return headers;
}
};
jsonObjectRequest.setRetryPolicy( new DefaultRetryPolicy( 3000,
    ↪ DefaultRetryPolicy.DEFAULT_MAX_RETRIES, DefaultRetryPolicy.
    ↪ DEFAULT_BACKOFF_MULT ) );
CustomRequestQueue.getInstance( getContext() ).addToRequestQueue(
    ↪ jsonObjectRequest);
}

private boolean turnBluetooth (boolean bluetooth){
if (bluetooth == true) {
// turn bluetooth off
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.
    ↪ getDefaultAdapter();
if (mBluetoothAdapter.isEnabled()) {
}else{
mBluetoothAdapter.enable();
}
return true;
} else {
// turn bluetooth off
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.
    ↪ getDefaultAdapter();
if (mBluetoothAdapter.isEnabled()) {
mBluetoothAdapter.disable();
}
return false;
}
}

@Override
public void onResume() {
super.onResume();
Profile profile = Profile.getCurrentProfile();
if (profile != null) {
getActivity().setTitle( "Home" );
userAccountFragment = UserAccountFragment.newInstance( profile );
mTextView.setText( "Welcome " + profile.getFirstName() +" "+ profile.
    ↪ getLastName() );
}
}

private void showDialog() {
if (!mProgressDialog.isShowing())
mProgressDialog.show();
}

private void hideDialog() {
if (mProgressDialog.isShowing())
mProgressDialog.dismiss();
}
}
```

16. Add to `fragment_main.xml` layout:

Listing B.12: fragment_main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
    ↪ android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".fragments.MainFragment">

<!-- TODO: Update blank fragment layout -->

<TextView
android:id="@+id/user_detail"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_gravity="center_horizontal|top"
android:layout_marginTop="30dp"
android:textSize="30dp"
android:textAlignment="center"/>
<RelativeLayout
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_gravity="center"
android:layout_centerVertical="true"
android:layout_centerHorizontal="true"
android:id="@+id/relativeLayout">
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Common Friends"
android:id="@+id/button_commonfriends"
android:textAllCaps="false"
android:width="180dp"
android:minHeight="20dp"
android:textSize="15dp"
android:textStyle="bold"
android:layout_gravity="center_horizontal"/>
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Account"
android:id="@+id/button_account1"
android:textAllCaps="false"
android:width="180dp"
android:minHeight="20dp"
android:textSize="15dp"
android:textStyle="bold"
android:layout_gravity="center_horizontal"
android:layout_below="@+id/button_commonfriends"/>
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Turn Discovery On/Off"
android:id="@+id/button_turndiscovery"
android:textAllCaps="false"
android:width="180dp"
```

```
android:minHeight="20dp"
android:textSize="15dp"
android:textStyle="bold"
android:layout_gravity="center_horizontal"
android:layout_below="@id/button_account1"/>
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Logout"
android:id="@+id/button_logout"
android:textAllCaps="false"
android:width="180dp"
android:minHeight="20dp"
android:textSize="15dp"
android:textStyle="bold"
android:layout_gravity="center_horizontal"
android:layout_below="@id/button_turndiscovery"/>
</RelativeLayout>
</RelativeLayout>
```

17. Add a new Fragment `UserAccountFragment`:

Listing B.13: UserAccountFragment

```
package com.example.xgc4811.myapp.fragments;

import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

import com.example.xgc4811.myapp.R;
import com.example.xgc4811.myapp.helper.AppHelper;
import com.facebook.Profile;
import com.facebook.login.widget.ProfilePictureView;

import java.util.HashMap;


public class UserAccountFragment extends Fragment {

private TextView mUserName;
private TextView mUserToken;
private ProfilePictureView profilePictureView;
private AppHelper mAppHelper;
public UserAccountFragment() {
// Required empty public constructor
}

@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
getActivity().setTitle( "User Account Detail" );
```

```
}

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
return inflater.inflate( R.layout.fragment_useraccount, container,
    ↪ false);
}

@Override
public void onViewCreated(View view, Bundle savedInstanceState) {
super.onViewCreated(view, savedInstanceState);
init( view );
}

public void init(View view){
mUserName = (TextView) view.findViewById(R.id.userName );
mUserToken = (TextView) view.findViewById( R.id.userToken );
profilePictureView = (ProfilePictureView) view.findViewById(R.id.
    ↪ profile_picture_view);
profilePictureView.setProfileId(getArguments().getString("image"));
mAppHelper = new AppHelper( getContext() );

mUserName.setText(getArguments().getString("name"));
// get user data from session
HashMap<String, String> user = mAppHelper.getUserDetails();
mUserToken.setText( user.get( AppHelper.USER_TOKEN ) );
}

public static UserAccountFragment newInstance (Profile profile){
UserAccountFragment user_account_frag = new UserAccountFragment();
Bundle args = new Bundle();
args.putString("name", profile.getName());
args.putString("image", profile.getId());
user_account_frag.setArguments(args);
return  user_account_frag;
}
}
```

18. Add to `fragment_useraccount.xml` layout:

Listing B.14: fragment_useraccount

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/
    ↪ android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context=".fragments.UserAccountFragment">

<!-- TODO: Update blank fragment layout -->
<com.facebook.login.widget.ProfilePictureView
android:id="@+id/profile_picture_view"
android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"/>
<TextView
android:id="@+id/userName"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/hello_blank_fragment"
android:textSize="20dp"
android:layout_toEndOf="@+id/profile_picture_view"
android:layout_marginTop="20dp"
android:layout_marginLeft="20dp"/>
<TextView
android:id="@+id/lblToken"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="Token:"
android:textSize="20dp"
android:layout_below="@+id/profile_picture_view"
android:layout_marginTop="20dp"/>
<TextView
android:id="@+id/userToken"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:text="@string/hello_blank_fragment"
android:textSize="20dp"
android:layout_below="@+id/lblToken"/>
</RelativeLayout>
```

19. Add a new Fragment `CommonFriends`:

Listing B.15: CommonFriends

```
package com.example.xgc4811.myapp.fragments;

import android.os.Bundle;
import android.support.v4.app.ListFragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import com.example.xgc4811.myapp.R;
import com.example.xgc4811.myapp.helper.AppHelper;
import com.example.xgc4811.myapp.services.BluetoothService;

import java.util.HashMap;
import java.util.Set;

public class CommonFriendsFragment extends ListFragment {

ArrayAdapter<String> listAdapter;
ListView mListView ;

AppHelper mAppHelper;
```

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
// Inflate the layout for this fragment
return inflater.inflate(R.layout.fragment_common_friends, container,
    ↪ false);
}

@Override
public void onViewCreated(View view, Bundle savedInstanceState) {
super.onViewCreated(view, savedInstanceState);
getActivity().setTitle( "Common Friends List" );
mListView = (ListView) view.findViewById( android.R.id.list );
listAdapter = new ArrayAdapter<String>( getActivity(), android.R.
    ↪ layout.simple_list_item_1,0 );
mListView.setAdapter(listAdapter);

mAppHelper = new AppHelper(getContext());

HashMap<String, Set<String>> commonfriends = mAppHelper.
    ↪ getCommonFriends();
try{
if ( commonfriends.size() > 0) {
Set<String> c = commonfriends.get(AppHelper.COMMON_FRIENDS);

for (String u : c) {
listAdapter.add(u);
}

}
}catch (NullPointerException e){
e.fillInStackTrace();
}
}
}
```

20. Add to `fragment_common_friends.xml` layout:

Listing B.16: fragment_common_friends.xml

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android
    ↪ "
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
tools:context="com.example.xgc4811.myapp.fragments.
    ↪ CommonFriendsFragment">
<!-- TODO: Update blank fragment layout -->
<ListView
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:id="@android:id/list"/>
</FrameLayout>
```

21. Add a new Java Interface `Constants`:

Listing B.17: Constants

```
package com.example.xgc4811.myapp;

/**
* Created by bakhs on 14/07/2016.
*/
public interface Constants {
// Message types sent from the BluetoothChatService Handler
public static final int MESSAGE_STATE_CHANGE = 1;
public static final int MESSAGE_READ = 2;
public static final int MESSAGE_WRITE = 3;
public static final int MESSAGE_DEVICE_NAME = 4;
public static final int MESSAGE_TOAST = 5;
// Key names received from the BluetoothChatService Handler
public static final String DEVICE_NAME = "device_name";
public static final String TOAST = "toast";
}
```

22. Add a new Java Class `MyApplication`, this class printout Facebook ID:

Listing B.18: MyApplication

```
package com.example.xgc4811.myapp;

import android.app.Application;
import android.content.Intent;
import android.content.pm.PackageInfo;
import android.content.pm.PackageManager;
import android.content.pm.Signature;
import android.util.Base64;
import android.util.Log;

import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;

/**
* Created by bakhs on 2/07/2016.
*/
public class MyApplication extends Application {
private static final String TAG = "MyApplication";

@Override
public void onCreate() {
super.onCreate();
// Add code to print out the key hash
printHash();
}

public void printHash(){
try {
```

```
PackageInfo info = getPackageManager().getPackageInfo(
"com.example.xgc4811.myapp",
PackageManager.GET_SIGNATURES);
for (Signature signature : info.signatures) {
MessageDigest md = MessageDigest.getInstance("SHA");
md.update(signature.toByteArray());
Log.d("KeyHash:", Base64.encodeToString(md.digest(), Base64.DEFAULT));
}
} catch (PackageManager.NameNotFoundException e) {

} catch (NoSuchAlgorithmException e) {
}
}
}
```

23. Add a new Java Class `AppHelper`, this class determines whether the user is login or not:

Listing B.19: AppHelper

```
package com.example.xgc4811.myapp.helper;

import android.content.Context;
import android.content.SharedPreferences;
import android.net.ConnectivityManager;

import java.util.HashMap;
import java.util.Set;

/**
* Created by bakhs on 23/07/2016.
*/
public class AppHelper {
// Shared Preferences
private SharedPreferences pref;

// Editor for Shared preferences
private SharedPreferences.Editor editor;

// Context
private Context mContext;

// Shared pref mode
private static final int PRIVATE_MODE = 0;

// Sharedpref file name
private static final String PREF_NAME = "CommonFriendPref";

// All Shared Preferences Keys
private static final String IS_LOGIN = "IsLoggedIn";

// User name (make variable public to access from outside)
public static final String USER_TOKEN = "token";
```

```
public static final String BLUETOOTH_ADDRESS = "BluetoothAddress";

public static final String COMMON_FRIENDS = "CommonFriends";

public AppHelper(Context context){
this.mContext = context;
pref = mContext.getSharedPreferences(PREF_NAME, PRIVATE_MODE);
editor = pref.edit();
}

/**
* Create login session
* */
public void createLoginSession(String token){
// Storing login value as TRUE
editor.putBoolean(IS_LOGIN, true);
// Storing token in pref
editor.putString( USER_TOKEN, token);
// commit changes
editor.commit();
}

/**
* Get stored session data
* */
public HashMap<String, String> getUserDetails(){
HashMap<String, String> user = new HashMap<String, String>();
// user name
user.put(USER_TOKEN, pref.getString(USER_TOKEN, null));
user.put( BLUETOOTH_ADDRESS, pref.getString( BLUETOOTH_ADDRESS, null )
    ↪ );
// return user
return user;
}

/**
* Clear session details
* */
public void logoutUser(){
// Clearing all data from Shared Preferences
editor.clear();
editor.commit();
}

/**
* Quick check for login
* **/
// Get Login State
public boolean isLoggedIn(){
return pref.getBoolean(IS_LOGIN, false);
}

public boolean isInternetAvailable(){
ConnectivityManager cm = (ConnectivityManager) mContext.
    ↪ getSystemService(mContext.CONNECTIVITY_SERVICE);
return cm.getActiveNetworkInfo() != null;
}
```

```
public void setDeviceAddress (String bluetoothAddress){
editor.putString( BLUETOOTH_ADDRESS,  bluetoothAddress);
editor.commit();
}

public void setCommonFriends(Set<String> commonFriends){
editor.putStringSet(COMMON_FRIENDS, commonFriends);
editor.commit();
}

public HashMap<String, Set<String>> getCommonFriends (){
HashMap<String, Set<String>> user = new HashMap<String, Set<String>>()
    ↪ ;
user.put(COMMON_FRIENDS, pref.getStringSet(COMMON_FRIENDS, null));


return user;

}
}
```

24. Add a new Java Class `CustomRequestQueue`, this class manages HTTP requests:

Listing B.20: CustomRequestQueue

```
package com.example.xgc4811.myapp.helper;

import android.content.Context;
import android.text.TextUtils;

import com.android.volley.Request;
import com.android.volley.RequestQueue;
import com.android.volley.toolbox.Volley;

/**
* Created by bakhs on 3/07/2016.
*/
public class CustomRequestQueue {
private static final String TAG = "CustomRequestQueue";
private static CustomRequestQueue mInstance;
private static Context mCtx;
private RequestQueue mRequestQueue;

public CustomRequestQueue(Context context) {
mCtx = context;
mRequestQueue = getRequestQueue();
}

public static synchronized CustomRequestQueue getInstance(Context
    ↪ context){
if (mInstance == null){
mInstance = new CustomRequestQueue(context);
```

```
}
return mInstance;
}

public RequestQueue getRequestQueue() {
if (mRequestQueue == null){
mRequestQueue = Volley.newRequestQueue(mCtx);
}
return mRequestQueue;
}

public <T> void addToRequestQueue(Request<T> req, String tag) {
req.setTag( TextUtils.isEmpty(tag) ? TAG : tag);
getRequestQueue().add(req);
}

public <T> void addToRequestQueue(Request<T> req) {
req.setTag(TAG);
getRequestQueue().add(req);
}

public void cancelPendingRequests(Object tag) {
if (mRequestQueue != null) {
mRequestQueue.cancelAll(tag);
}
}
}
```

25. Add a Java Class `NetworksReceiver`, this class helps to determine if the user is connected to an internet:

Listing B.21: NetworksReceiver

```
package com.example.xgc4811.myapp.receivers;

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.net.ConnectivityManager;
import android.util.Log;
import android.widget.Toast;

import com.example.xgc4811.myapp.services.BluetoothService;
import com.example.xgc4811.myapp.helper.AppHelper;

/**
* Created by bakhs on 20/07/2016.
*/
public class NetworksReceiver extends BroadcastReceiver {
private static final String TAG="NetworksReceiver";

@Override
```

```
public void onReceive(Context context, Intent intent) {
String action = intent.getAction();
Log.d("BroadcastActions", "Action "+action+" received");
int state;
BluetoothDevice bluetoothDevice;
AppHelper mAppHelper = new AppHelper( context );
switch(action)
{
case BluetoothAdapter.ACTION_STATE_CHANGED:
state = intent.getIntExtra(BluetoothAdapter.EXTRA_STATE, -1);
if (state == BluetoothAdapter.STATE_OFF)
{
context.stopService( new Intent( context, BluetoothService.class ) );
Toast.makeText(context, "Bluetooth is off", Toast.LENGTH_SHORT).show()
    ↪ ;
Log.d("BroadcastActions", "Bluetooth is off");
}
else if(state == BluetoothAdapter.STATE_ON)
{
Log.d("BroadcastActions", "Bluetooth is on");
}
break;

case BluetoothDevice.ACTION_ACL_CONNECTED:
bluetoothDevice = intent.getParcelableExtra(BluetoothDevice.
    ↪ EXTRA_DEVICE);
Toast.makeText(context, "Connected to "+ bluetoothDevice.getName(),
Toast.LENGTH_SHORT).show();
Log.d("BroadcastActions", "Connected to "+ bluetoothDevice.getName());
break;

case BluetoothDevice.ACTION_ACL_DISCONNECTED:
bluetoothDevice = intent.getParcelableExtra(BluetoothDevice.
    ↪ EXTRA_DEVICE);
Toast.makeText(context, "Disconnected from "+bluetoothDevice.getName()
    ↪ ,
Toast.LENGTH_SHORT).show();
break;
}

ConnectivityManager cm = (ConnectivityManager) context.
    ↪ getSystemService(Context.CONNECTIVITY_SERVICE);
if (cm.getActiveNetworkInfo() == null){
Toast.makeText( context,"No Internet Connection", Toast.LENGTH_SHORT )
    ↪ .show();
}
}
}
```

26. Add a new Service Class `BluetoothService`, this class enables the application to do some activities in the background, where the application uses Bluetooth to search for nearby devices. In addition, after receiving a response from the server, it starts calculating the distance between the devices. Finally, it informs

the users about the results through a notification.

```
package com.example.xgc4811.myapp.services;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.os.Handler;
import android.os.IBinder;
import android.os.Message;
import android.support.annotation.Nullable;
import android.support.v4.app.NotificationCompat;
import android.util.Log;

import com.android.volley.AuthFailureError;
import com.android.volley.DefaultRetryPolicy;
import com.android.volley.Request;
import com.android.volley.Response;
import com.android.volley.VolleyError;
import com.android.volley.toolbox.JsonObjectRequest;
import com.example.xgc4811.myapp.helper.CustomRequestQueue;
import com.example.xgc4811.myapp.R;
import com.example.xgc4811.myapp.activities.ResultsActivity;
import com.example.xgc4811.myapp.helper.AppHelper;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Random;
import java.util.Set;
import java.util.Timer;
import java.util.TimerTask;

/**
* Created by bakhs on 20/07/2016.
*/
public class BluetoothService extends Service {

private static final String TAG = "BluetoothService";
private static final String BLUETOOTH_SEARCH_URL = "http://bakhshb.
    ↪ pythonanywhere.com/api/bluetooth/search/";
// Handler
private static final int SEND_REQUEST = 1;
private static final int SEND_RESULTS_ALREADY_FRIEND = 2;
private static final int SEND_RESULTS_MUTUAL_FRIEND = 3;
```

```
private static final int SEND_RESULTS_SUGGEST_FRIEND=4;

//Response friend status
private static final int NO_MATCH = 0;
private static final int ALREADY_FRIEND = 1;
private static final int FOUND_MUTUAL = 2;
private static final int SUGGEST_FRIEND=3;

private BluetoothAdapter mBluetoothAdapter;
private IntentFilter mIntentFilter;
private AppHelper mAppHelper;

private Set<String> devicesCommonFriends;

private Set<String> foundDevices;

@Nullable
@Override
public IBinder onBind(Intent intent) {
return null;
}

@Override
public void onCreate() {
super.onCreate();
mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
mAppHelper = new AppHelper( getApplicationContext() );
devicesCommonFriends = new HashSet<String>(  );
foundDevices = new HashSet<String>(  );

//Send Notification every 15 min when device is found, otherwise keeps
    ↪  search every 3 mins
Timer t = new Timer();
t.schedule( new TimerTask() {
@Override
public void run() {
foundDevices.clear();
}
},0,900000 );
}

@Override
public int onStartCommand(Intent intent, int flags, int startId) {
Log.i( TAG, "onStartCommand: Bluetooth Service Started" );
if (mBluetoothAdapter.isDiscovering()) {
mBluetoothAdapter.cancelDiscovery();
}
mBluetoothAdapter.startDiscovery();
mIntentFilter = new IntentFilter( BluetoothDevice.ACTION_FOUND );

getApplicationContext().registerReceiver(mBroadcastReceiver,
    ↪ mIntentFilter );
return super.onStartCommand( intent, flags, startId );
}

@Override
public void onDestroy() {
```

```
super.onDestroy();
getApplicationContext().unregisterReceiver( mBroadcastReceiver );
NotificationManager mNotificationManager = mNotificationManager();
mNotificationManager.cancelAll();
Log.i( TAG, "onDestroy: Bluetooth Service Stopped" );
}

private final BroadcastReceiver mBroadcastReceiver = new
    ↪ BroadcastReceiver() {
@Override
public void onReceive(Context context, Intent intent) {
String action = intent.getAction();
BluetoothDevice device = null;
if (BluetoothDevice.ACTION_FOUND.equals( action )){
device= intent.getParcelableExtra( BluetoothDevice.EXTRA_DEVICE );
int  rssi = intent.getShortExtra(BluetoothDevice.EXTRA_RSSI,Short.
    ↪ MIN_VALUE);
// Devices not paired
if (device.getBondState() != BluetoothDevice.BOND_BONDED) {
// Devices found in the server
if (!foundDevices.contains( device.getAddress() )) {
Log.i( TAG, "onReceive:  RSSI: " + rssi + "dBm " + device.getAddress()
    ↪  + " " + device.getName() );
Message mMessage = mHandler.obtainMessage( SEND_REQUEST );
Bundle mBundle = new Bundle();
mBundle.putString( "bluetooth_address", device.getAddress() );
mBundle.putInt( "rssi", rssi );
mMessage.setData( mBundle );
mHandler.sendMessage( mMessage );
}

}
}
}
};


private void sendNotification(String msg) {
NotificationManager mNotificationManager = mNotificationManager();
Intent mIntent = new Intent( this.getApplicationContext(),
    ↪ ResultsActivity.class );
Bundle mBundle = new Bundle(  );
mBundle.putString( "msg", msg );
mIntent.putExtras( mBundle );


PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
mIntent, PendingIntent.FLAG_UPDATE_CURRENT);

NotificationCompat.Builder mBuilder =
new NotificationCompat.Builder(this)
.setSmallIcon( R.drawable.ic_stat_name)
.setContentTitle("Bluetooth Device Next To You")
.setStyle(new NotificationCompat.BigTextStyle()
.bigText(msg))
.setContentText(msg)
.setDefaults( Notification.DEFAULT_SOUND ).setAutoCancel( true );
```

```
mBuilder.setContentIntent(contentIntent);
Random random = new Random();
int m = random.nextInt(9999 - 1000) + 1000;
mNotificationManager.notify(m, mBuilder.build());
}

private NotificationManager mNotificationManager (){
return (NotificationManager) this.getSystemService(Context.
    ↪ NOTIFICATION_SERVICE);
}

// calculate bluetooth distance
private String calculateDistance (int rssi){
if (rssi > -45){
return "0 Meter";
} else if (-45 > rssi && rssi >= -47){
return "1 Meter";
} else if (-47 > rssi && rssi >= -51){
return "2 Meters";
} else if(-51 > rssi && rssi >= -54){
return "3 Meters";
} else if (-54 > rssi && rssi >= -58){
return "4 Meters";
} else if (-58 > rssi && rssi >= -61){
return "5 Meters";
}
return "more than 5 Meters";
}

private void bluetoothSearchRequest(final String bluetooth_address,
    ↪ final int rssi){
HashMap<String, String > user = mAppHelper.getUserDetails();
final String token = user.get( AppHelper.USER_TOKEN );
final String currentDevice = user.get( AppHelper.BLUETOOTH_ADDRESS );
//
HashMap<String, String> params = new HashMap<String, String>();
params.put("bluetooth", bluetooth_address);
JsonObjectRequest jsonObjectRequest = new JsonObjectRequest( Request.
    ↪ Method.POST, BLUETOOTH_SEARCH_URL, new JSONObject( params ), new
    ↪  Response.Listener<JSONObject>() {
@Override
public void onResponse(JSONObject response) {
try {
foundDevices.add( bluetooth_address );
int count = 0;
Log.d( TAG, "onResponse: " + response.getString( "status" ) +" " +
    ↪ response.getInt( "friend_status") );
if (response.getInt( "friend_status" ) == ALREADY_FRIEND){
Message mMessage = mHandler.obtainMessage(SEND_RESULTS_ALREADY_FRIEND)
    ↪ ;
Bundle mBundle = new Bundle(  );
mBundle.putString( "user", response.getString( "user" ) );
mBundle.putInt( "rssi", rssi );
mMessage.setData( mBundle );
mHandler.sendMessage( mMessage );
}else if (response.getInt( "friend_status" ) == FOUND_MUTUAL) {
JSONArray mJsonArray = response.getJSONArray( "friend" );
```

```
for (int i = 0; i < mJsonArray.length(); i++) {
count++;
}

Message mMessage = mHandler.obtainMessage(SEND_RESULTS_MUTUAL_FRIEND);
Bundle mBundle = new Bundle(  );
mBundle.putString( "user", response.getString( "user" ) );
mBundle.putInt( "count", count );
mBundle.putInt( "rssi", rssi );
mMessage.setData( mBundle );
mHandler.sendMessage( mMessage );
if (!devicesCommonFriends.contains(bluetooth_address)) {
devicesCommonFriends.add("Found " + count + " Mutual Friend with
    ↪ Bluetooth Address " + bluetooth_address + " Name " + response.
    ↪ getString("user"));
mAppHelper.setCommonFriends(devicesCommonFriends);
}
} else if (response.getInt( "friend_status" ) == SUGGEST_FRIEND){
Message mMessage = mHandler.obtainMessage(SEND_RESULTS_SUGGEST_FRIEND)
    ↪ ;
Bundle mBundle = new Bundle(  );
mBundle.putString( "user", response.getString( "user" ) );
mBundle.putInt( "rssi", rssi );
mMessage.setData( mBundle );
mHandler.sendMessage( mMessage );
}


} catch (JSONException e) {
e.printStackTrace();
}
}
}, new Response.ErrorListener() {
@Override
public void onErrorResponse(VolleyError error) {
Log.d( TAG, "onErrorResponse: " +  bluetooth_address);
}
} ){
@Override
public Map<String, String> getHeaders() throws AuthFailureError {
HashMap<String, String> headers = new HashMap<String, String>();
headers.put("Authorization","Token "+ token);
headers.put("Content-Type","application/json");
return headers;
}
};
jsonObjectRequest.setRetryPolicy( new DefaultRetryPolicy( 3000,
    ↪ DefaultRetryPolicy.DEFAULT_MAX_RETRIES, DefaultRetryPolicy.
    ↪ DEFAULT_BACKOFF_MULT ) );
CustomRequestQueue.getInstance( getApplicationContext() ).
    ↪ addToRequestQueue(jsonObjectRequest);
}

private final Handler mHandler = new Handler(  ){
@Override
public void handleMessage(Message msg) {
String user = null;
```

```
int rssi = 0;
switch (msg.what){
case SEND_REQUEST:
bluetoothSearchRequest( msg.getData().getString( "bluetooth_address" )
    ↪ , msg.getData().getInt( "rssi" ) );
break;
case SEND_RESULTS_ALREADY_FRIEND:
user = msg.getData().getString( "user" );
rssi = msg.getData().getInt( "rssi" );
Log.d( TAG, "handleMessage: " + rssi );
Log.d( TAG, "handleMessage: " + user );
sendNotification( "Your Friend "+ user + "\n Proximate Distance: " +
    ↪ calculateDistance( rssi ) );
break;
case SEND_RESULTS_MUTUAL_FRIEND:
user = msg.getData().getString( "user" );
int count = msg.getData().getInt( "count" );
rssi = msg.getData().getInt( "rssi" );
Log.d( TAG, "handleMessage: " + rssi );
Log.d( TAG, "handleMessage: " + user );
if (count <=1) {
sendNotification( "You have  " + count + " mutual friend with " + user
    ↪  + "\n Proximate Distance: " + calculateDistance( rssi ) );
}else{
sendNotification( "You have  " + count + " mutual friends with " +
    ↪ user + "\n Proximate Distance: " + calculateDistance( rssi ) );
}
break;
case SEND_RESULTS_SUGGEST_FRIEND:
user = msg.getData().getString( "user" );
rssi = msg.getData().getInt( "rssi" );
Log.d( TAG, "handleMessage: " + rssi );
Log.d( TAG, "handleMessage: " + user );
sendNotification( "Friend Suggestion "+ user + "\n Proximate Distance:
    ↪  " + calculateDistance( rssi ) );
break;
}
}
};

}
```

27. Deploy to an actual Android Device, because the emulator does not allow to use Bluetooth, which is required in this project.