Visvesvaraya Technological University

Jnana Sangama, Belagavi - 590018

A Project Report on

"Intelligent Messaging Assistant (IMA)"

Project Report submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING

Submitted by

NIKHIL GUPTA M. C.	**1KS13CS065**
NIKHIL N. PRASAD	**1KS13CS066**
VINAYAKA K.	**1KS13CS120**
VIVEK B.	**1KS13CS122**

Under the guidance of
Mr. HARSHAVARDHAN J. R.
Associate Professor
Department of Computer Science & Engineering
K.S.I.T. Bengaluru-560109

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
K. S. Institute of Technology
#14, Raghuvanahalli, Kanakapura Road, Bengaluru - 560109

2016 - 2017

K. S. Institute of Technology

#14, Raghuvanahalli, Kanakapura Road, Bengaluru - 560109

Department of Computer Science & Engineering

CERTIFICATE

Certified that the project work entitled **"Intelligent Messaging Assistant (IMA)"** is a bonafide work carried out by:

NIKHIL GUPTA M. C.	**1KS13CS065**
NIKHIL N. PRASAD	**1KS13CS066**
VINAYAKA K.	**1KS13CS120**
VIVEK B.	**1KS13CS122**

in partial fulfillment for VIII semester B.E., Project Work in the branch of Computer Science and Engineering prescribed by **Visvesvaraya Technological University, Belagavi** during the period of February 2017 to June 2017. It is certified that all the corrections and suggestions indicated for internal assessment have been incorporated in the report deposited in the department library. The Project Report has been approved as it satisfies the academic requirements in report of project work prescribed for the Bachelor of Engineering degree.

.....	.....	.....
Signature of the Guide	**Signature of the HOD**	**Signature of the Principal/Director**
(Mr. Harshavardhan J. R.)	(Dr. Rekha B. Venkatapur)	(Dr. T. V. Govindaraju)

External Viva

Name of the Examiners	Signature with date
1.	
2.	

DECLARATION

We, the undersigned students of 8th semester, Computer Science & Engineering, KSIT, declare that our project work titled "Intelligent Messaging Assistant (IMA)", is a bonafide work of ours. Our project is neither a copy nor by means a modification of any other engineering project.

We also declare that this project was not entitled for submission to any other university in the past and shall remain the only submission made and will not be submitted by us to any other university in the future.

Place:

Date:

Name	USN	Signature
NIKHIL GUPTA M. C.	1KS13CS065	……..
NIKHIL N. PRASAD	1KS13CS066	……..
VINAYAKA K.	1KS13CS120	……..
VIVEK B.	1KS13CS122	……..

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task will be incomplete without the mention of the individuals, we are greatly indebted to, who through guidance and providing facilities have served as a beacon of light and crowned our efforts with success.

First and foremost, our sincere prayer goes to almighty, whose grace made us realize our objective and conceive this project. We take pleasure in expressing our profound sense of gratitude to our parents for helping us complete our project work successfully.

We take this opportunity to express our sincere gratitude to our college **K.S. Institute of Technology, Bengaluru** for providing a very good infrastructure and the environment to work for the project.

We would like to express our gratitude to **Dr. T. V. Govindaraju,** Principal/Director, for providing all the kindness forwarded to us in carrying out this project work in college.

We like to extend our gratitude to **Dr. Rekha B. Venkatapur**, Professor and Head of the Department, for supporting us in carrying out this project and for allowing us to use library and lab facilities.

Also, we are thankful to **Mr. Harshavardhan J. R.,** Associate Professor, for being our project guide, under whose able guidance this project work has been carried out and completed successfully.

We also like to thank our Project Coordinators, **Mr. Pradeep Kumar G. H.**, Assistant Professor, and **Mr. Kumar K.**, Assistant Professor for their help and support provided to carry out the project and complete it successfully.

We are also thankful to the teaching and non - teaching staff of Computer Science & Engineering, KSIT for helping us in completing this project work.

ABSTRACT

One of the most common problems we face every day is that, people often forget minor, but crucial jobs we were requested to perform, such as calling someone at a later time, mailing an important document etc. People who forget to do so often feel dissatisfied that they did not perform the assigned task. The aim of this project is to create a messaging application that recognizes the context of its users' conversation, stores it, analyses the conversations, determines the Action-Driving Statements, and triggers a reminder to its user, thereby informing him/her to carry out the task decided during the conversation. Integration and possibilities of the IMA in real world scenarios are considered and described in this paper. The possible design and implementation of an autonomous reminder and conversation summarizer is presented and described. The development frameworks as well as technologies that are used to realize this concept is described. The concept of text summarization and detection of plausible reminders implementable using machine learning is also described.

Keywords: Automatic conversation summarizer, Instant Messaging (IM), XMPP, Natural Language Processing, Machine Learning.

CONTENTS

CONTENTS

CONTENTS

LIST OF FIGURES

LIST OF SNAPSHOTS

LIST OF TABLES

LIST OF ABBREVATIONS

Abbreviation	Description
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
IDEN	Integrated Digital Enhanced Network
IM	Instant Messaging
IMA	Intelligent Messaging Application
LUIS	Language Understanding Intelligent System
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NLU	Natural Language Understanding
REST	Representational State Transfer
TCP	Transmission Control Protocol
UML	Unified Modelling Language
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

INTRODUCTION

CHAPTER 1

INTRODUCTION

The advancement of the Internet and the smartphone have changed the way people communicate in this era. One of the biggest advancements due to the advent of these technologies is that most conversations today happen through electronic mail and instant messaging. Instant messaging is a set of communication technologies used for text-based real-time communication between two or more participants over the Internet. Often referred to as "chat", IM offers real-time communication between two or more users. IM solutions, today, include group chat, conferencing, voice, video, and much more. IM is highly popular with both consumer and enterprise users. This new trend of conversing through Instant messaging has opened up an opportunity to analyse the conversation data and build useful applications using them. One such application of conversation analysis is devising a solution to the aforementioned problem. In recent years, the number of smartphone users have surpassed the number of PC users, thanks to the portability and convenience of prompt information access. This has encouraged advancements in mobile computing and the popularity of mobile operating systems such as Android. Since this advent of smartphones, the general messaging paradigm has shifted drastically to these handheld devices. worldwide IM user accounts are expected to grow from over 5.8 billion in 2017 to over 8.3 billion by year-end 2021, representing an average annual growth rate of 9% Every smartphone users uses IM and it is found that Mobile users spent more than 10% of their mobile usage time on Social Messaging Apps Stephen Wang, a senior PM at WeChat said "At every time throughout the day, there is a touchpoint between WeChat and your normal life." Thus, a large portion of casual communication happens through IM on a daily basis, which often contains sentences that assigns a job responsibility to either of the communication participants. For example, in a conversation between students, student ***x*** may text student ***y*** to get document **z** to college. We often have a tendency to forget these tiny responsibilities, and thus, an automatic reminder about this responsibility would be helpful to user. IM also contains group chats. Typically, these chat rooms are always filled with conversations and not all chats are important to all the users. Thus, a summary of these conversations will ensure that the user will not miss out on any important messages in the group. For example, an IM group containing student participants, which would normally comprise much of

general conversations, would also have traces of messages regarding placements during placement season. In such a scenario, a student may fail to notice a vital placement intimation due to an abundance of mundane messages. This could prove to create a gulf in communication. One of the ways to bridge this gap is to automatically create a summary of the conversation, which provides the user with the important messages. This project aims in providing reminders for eventful messages and a conversation summary of instant messages. The software consists of a chat paradigm, meaning a user can communicate with another user via text message (similar to popular messaging services such as WhatsApp, Facebook Messenger, and so on). The improvement that our application boasts, is the reminder and summarizer feature. It keeps a detailed summary of the conversation between the users in real-time, and automatically detects the context of the conversation. The application is always on the look-out for anything that resembles an imperative sentence. Once it does, a reminder is added to the database maintained by the application, and a notification is set at the given time to the user. The user can also go through a history of all his/her reminders within the application itself. This describes the basic functionality of the proposed application.

1.1 NATURAL LANGUAGE PROCESSING (NLP)

Natural language processing (**NLP**) is a field of computer science, artificial intelligence and computational linguistics concerned with the interactions between computers and human (natural) languages, and, in particular, concerned with programming computers to fruitfully process large natural language corpora. Challenges in natural language processing frequently involve natural language understanding, natural language generation (frequently from formal, machine-readable logical forms), connecting language and machine perception, managing human-computer dialog systems, or some combination thereof.

NLP research has relied heavily on machine learning. Formerly, many language-processing tasks typically involved the direct hand coding of rules, which is not in general robust to natural language variation. The machine-learning paradigm calls instead for using statistical inference to automatically learn such rules through the analysis of large *corpora* of typical real-world examples (a *corpus* (plural, "corpora") is a set of documents, possibly with human or computer annotations).

Many different classes of machine learning algorithms have been applied to NLP

tasks. These algorithms take as input a large set of "features" that are generated from the input data. Some of the earliest-used algorithms, such as decision trees, produced systems of hard if-then rules similar to the systems of hand-written rules that were then common. Increasingly, however, research has focused on statistical models, which make soft, probabilistic decisions based on attaching real-valued weights to each input feature. Such models have the advantage that they can express the relative certainty of many different possible answers rather than only one, producing more reliable results when such a model is included as a component of a larger system.

Systems based on machine-learning algorithms have many advantages over hand-produced rules:

- The learning procedures used during machine learning automatically focus on the most common cases, whereas when writing rules by hand it is often not at all obvious where the effort should be directed.

- Automatic learning procedures can make use of statistical inference algorithms to produce models that are robust to unfamiliar input (e.g. containing words or structures that have not been seen before) and to erroneous input (e.g. with misspelled words or words accidentally omitted). Generally, handling such input gracefully with hand-written rules—or more generally, creating systems of hand-written rules that make soft decisions—is extremely difficult, error-prone and time-consuming.

- Systems based on automatically learning the rules can be made more accurate simply by supplying more input data. However, systems based on hand-written rules can only be made more accurate by increasing the complexity of the rules, which is a much more difficult task. In particular, there is a limit to the complexity of systems based on hand-crafted rules, beyond which the systems become more and more unmanageable. However, creating more data to input to machine-learning systems simply requires a corresponding increase in the number of man-hours worked, generally without significant increases in the complexity of the annotation process.

1.1.1 SENTENCE CLASSIFICTION

Classification is the task of choosing the correct **class label** for a given input. In basic classification tasks, each input is considered in isolation from all other inputs, and the set of labels is defined in advance. Some examples of classification tasks are:

- Deciding whether an email is spam or not.
- Deciding what the topic of a news article is, from a fixed list of topic areas such as "sports," "technology," and "politics."
- Deciding whether a given occurrence of the word *bank* is used to refer to a river bank, a financial institution, the act of tilting to the side, or the act of depositing something in a financial institution.

The basic classification task has a number of interesting variants. For example, in multi-class classification, each instance may be assigned multiple labels; in open-class classification, the set of labels is not defined in advance; and in sequence classification, a list of inputs is jointly classified.

In classification, labelled data typically consists of a bag of multidimensional feature vectors (normally called X) and for each vector a label, Y which is often just an integer corresponding to a category E.g. (face=1, non-face=-1). Unlabelled data misses the Y component. There are many scenarios where unlabelled data is plentiful and easily obtained but labelled data often requires a human/expert to annotate.

Typically, unlabelled data consists of samples of natural or human-created artefacts that you can obtain relatively easily from the world. Some examples of unlabelled data might include photos, audio recordings, videos, news articles, tweets, x-rays (if you were working on a medical application), etc. There is no "explanation" for each piece of unlabelled data -- it just contains the data, and nothing else.

Labelled data typically takes a set of unlabelled data and augments each piece of that unlabelled data with some sort of meaningful "tag," "label," or "class" that is somehow informative or desirable to know. For example, labels for the above types of unlabelled data might be whether this photo contains a horse or a cow, which words were uttered in this audio recording, what type of action is being performed in this video, what the topic of this

news article is, what the overall sentiment of this tweet is, whether the dot in this x-ray is a tumour, etc.

Labels for data are often obtained by asking humans to make judgments about a given piece of unlabelled data (e.g., "Does this photo contain a horse or a cow?") and are significantly more expensive to obtain than the raw unlabelled data.

After obtaining a labelled dataset, machine learning models can be applied to the data so that new unlabelled data can be presented to the model and a likely label can be guessed or predicted for that piece of unlabelled data.

A classifier is called **supervised** if it is built based on training corpora containing the correct label for each input. The framework used by supervised classification is shown in **Figure 1.1.**

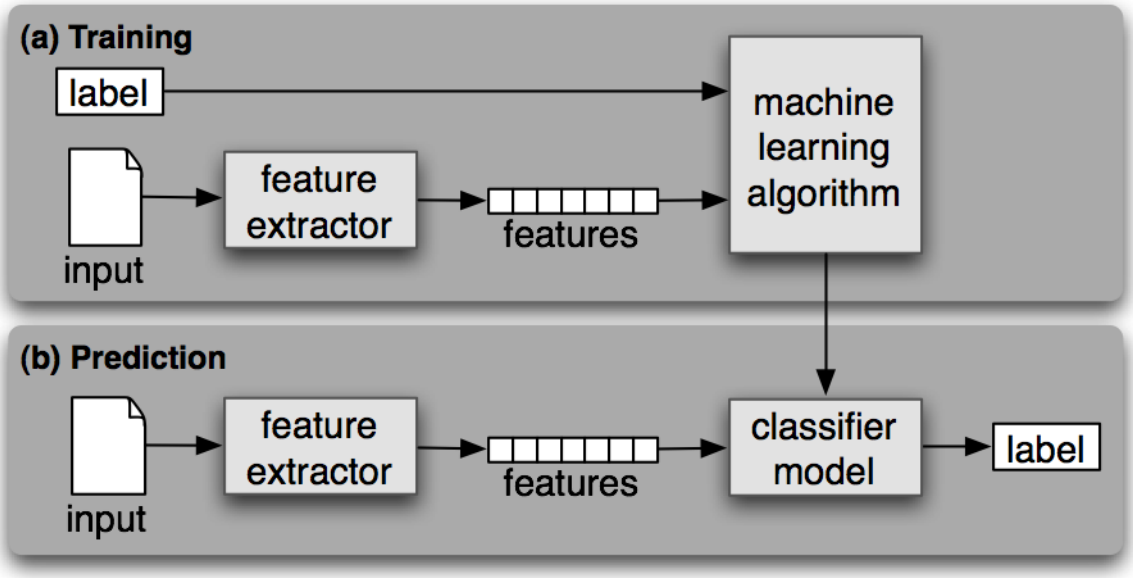

Figure 1.1: Supervised Classification

- **Training:** a feature extractor is used to convert each input value to a feature set. These feature sets, which capture the basic information about each input that should be used to classify it, are discussed in the next section. Pairs of feature sets and labels are fed into the machine learning algorithm to generate a model.

- **Prediction:** the same feature extractor is used to convert unseen inputs to feature sets. These feature sets are then fed into the model, which generates predicted labels.

1.2 MESSAGING APPLICATIONS

A messaging app is a pretty straightforward concept: they replace text messaging. They're especially useful for anyone who doesn't want to pay for texting (or is communicating across country borders).

Chances are, if you have a smartphone, you've already encountered a messaging app. If you're on Android, Google Hangouts is the default for messaging with other Android (and Google) users. If you're on the iPhone, the Messages app has texting and iMessage (non-SMS messages between iOS devices) built-in. If you want a simple way to communicate with people between different devices, third-party messaging client make things easy. That's where apps like WhatsApp or Snapchat comes in.

The appeal of these apps depends on the user. For some, it's a way to get around SMS fees. This is especially the case if you're communicating (or just travelling) internationally. For others, it's a way to keep conversations private. Other apps let you set your status to tell friends you're busy. All these make SMS more interesting by adding features, but which features you want depends on your needs.

The major issue with messaging apps is that, like a social network, each app is only as useful as the people using it. None of these apps can communicate outside of the app or between different apps. For example, if you want to use WhatsApp but none of your friends are using it, you can't really do much with it. You pretty much have to go with what your friends use. Messaging apps aren't for everyone, and if you're not an avid texter, your friends aren't using any of these services, or you're not worried about text messaging fees, none of the messaging apps are really useful. But if any of those things do spark your interest, they're worth investing a little time in to pick one you like.

1.3 XMPP

The Extensible Messaging and Presence Protocol (XMPP) is an open Extensible Markup Language (XML) protocol for near-real-time messaging, presence, and request-response services.

1.3.1 Overview

Although XMPP is not wedded to any specific network architecture, to date it usually has been implemented via a client-server architecture wherein a client

utilizing XMPP accesses a server over a TCP connection, and servers also communicate with each other over TCP connections.

The following diagram provides a high-level overview of this architecture (where "-" represents communications that use XMPP and "=" represents communications that use any other protocol).

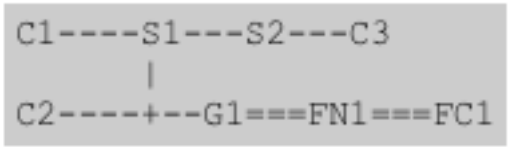

Figure 1.2: XMPP Architecture

The symbols are as follows:

- C1, C2, C3 = XMPP clients
- S1, S2 = XMPP servers
- G1 = A gateway that translates between XMPP and the protocol(s) used on a foreign (non-XMPP) messaging network
- FN1 = A foreign messaging network
- FC1 = A client on a foreign messaging network

1.3.2 Server

A server behaves as an intelligent abstraction layer for XMPP communications. Its primary responsibilities are:

- to manage connections from or sessions for other entities, in the form of **XML streams** to and from authorized clients, servers, and other entities
- to route appropriately-addressed **XML stanzas** among such entities over XML streams

Most XMPP-compliant servers also assume responsibility for the storage of data that is used by clients (e.g., contact lists for users of XMPP-based instant messaging and presence applications); in this case, the XML data is processed directly by the server itself on behalf of the client and is not routed to another entity.

1.3.3 Client

Most clients connect directly to a server over a TCP connection and use XMPP to take full advantage of the functionality provided by a server and any associated services. Multiple resources (e.g., devices or locations) MAY connect simultaneously to a server on behalf of each authorized client, with each resource differentiated by the resource identifier of an XMPP address (e.g., <node@domain/home> vs. <node@domain/work>) as defined under addressing scheme. The recommended port for connections between a client and a server is 5222, as registered with the IANA (Internet Assigned Numbers Authority).

1.3.4 Gateway

A gateway is a special-purpose server-side service whose primary function is to translate XMPP into the protocol used by a foreign (non-XMPP) messaging system, as well as to translate the return data back into XMPP. Examples are gateways to email, Internet Relay Chat, SIMPLE, Short Message Service (SMS), and legacy instant messaging services such as AIM, ICQ, MSN Messenger, and Yahoo! Instant Messenger. Communications between gateways and servers, and between gateways and the foreign messaging system, are not defined in this document.

1.3.5 Network

Because each server is identified by a network address and because server-to-server communications are a straightforward extension of the client-to-server protocol, in practice, the system consists of a network of servers that inter-communicate. Thus, for example, <juliet@example.com> is able to exchange messages, presence, and other information with <romeo@example.net>. This pattern is familiar from messaging protocols (such as SMTP) that make use of network addressing standards. Communications between any two servers are OPTIONAL. If enabled, such communications should occur over XML streams that are bound to TCP connections. The recommended port for connections between servers is 5269, as registered with the IANA.

REQUIREMENTS ANALYSIS

CHAPTER 2

REQUIREMENTS ANALYSIS

Requirements analysis, also called requirements engineering, is the process of determining user expectations for a new or modified product. These features, called requirements, must be quantifiable, relevant and detailed. In software engineering, such requirements are often called functional specification. Requirements analysis is an important aspect of project management.

2.1 SOFTWARE REQUIREMENTS

Software Requirements is a field within software engineering that deals with establishing the needs of stakeholders that are to be solved by software. In other words, it is a complete description of the behaviour of a system to be developed and may include a set of use cases that describes interaction of the user with the software. Following are the software requirements.

2.1.1 JAVA

Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere", meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture. Java was originally developed by James Gosling at Sun Microsystems (which has since been acquired by Oracle Corporation) and released in 1995 as a core component of Sun Microsystems' Java platform. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

One design goal of Java is portability, which means that programs written for the Java platform must run similarly on any combination of hardware and operating system with adequate runtime support. This is achieved by compiling the Java language code to an intermediate representation called Java bytecode, instead of directly to

architecture-specific machine code. Java bytecode instructions are analogous to machine code, but they are intended to be executed by a virtual machine (VM) written specifically for the host hardware. End users commonly use a Java Runtime Environment (JRE) installed on their own machine for standalone Java applications, or in a web browser for Java applets.

Standard libraries provide a generic way to access host-specific features such as graphics, threading, and networking.

The use of universal bytecode makes porting simple. However, the overhead of interpreting bytecode into machine instructions made interpreted programs almost always run more slowly than native executables. Just-in-time (JIT) compilers that compile bytecodes to machine code during runtime were introduced from an early stage. Java itself is platform-independent and is adapted to the particular platform it is to run on by a Java virtual machine for it, which translates the Java bytecode into the platform's machine language.

2.1.2 ANDROID

Android is a mobile operating system developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. Android's user interface is mainly based on direct manipulation, using touch gestures that loosely correspond to real-world actions, such as swiping, tapping and pinching, to manipulate on-screen objects, along with a virtual keyboard for text input.

Initially developed by Android Inc., which Google bought in 2005, Android was unveiled in 2007, along with the founding of the Open Handset Alliance – a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices.

Android's source code is released by Google under an open source license, although most Android devices ultimately ship with a combination of free and open source and proprietary software, including proprietary software required for accessing Google services. Android is popular with technology companies that require a ready-made, low-cost and customizable operating system for high-tech devices.

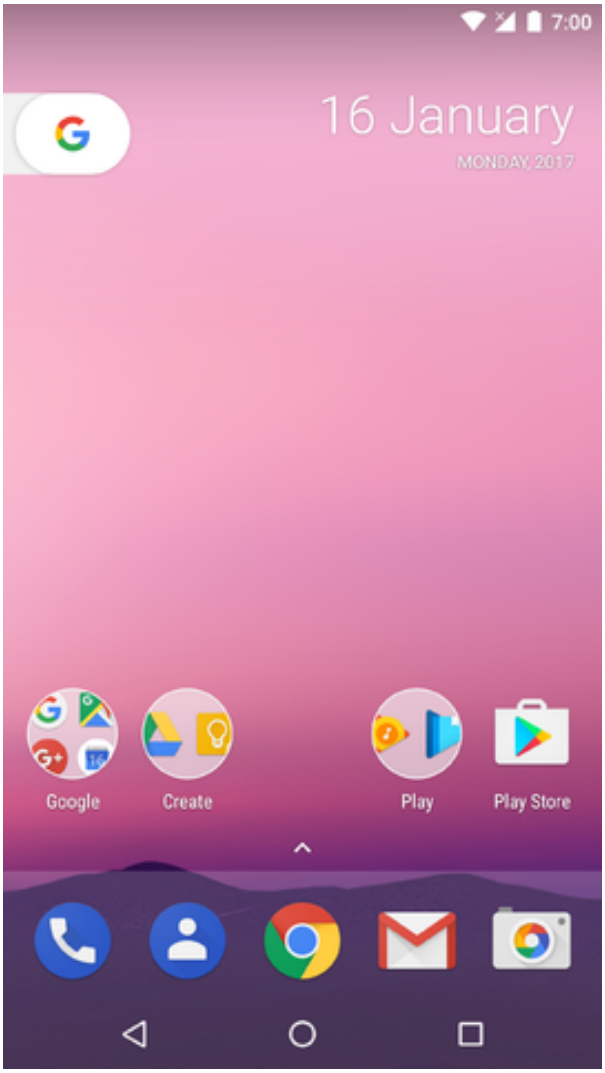

Figure 2.1: Android Nougat Home Screen

Features of Android:

- **Open Source:** Android is an open source operating system.
- **Handset layouts:** The platform is adaptable to larger, VGA, 2D graphics library,

  3D graphics library based on OpenGL ES 2.0 specifications, and traditional smart

  phone layouts.
- **Storage:** The Database Software SQLite is used for data storage purposes.
- **Connectivity:** Android supports connectivity technologies including GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, and WiMAX.

- **Web browser:** The web browser available in Android is based on the open-source
  Web Kit application framework. The browser scores a 93/100 on the Acid3 Test.
- **Java support:** Software written in Java can be compiled to be executed in the Dalvik virtual machine, which is a specialized VM implementation designed for
  mobile device use, although not technically a standard Java Virtual Machine. Android does not support J2ME, like some other mobile operating systems.
- **Media support:** Android supports the following audio/video/still media formats:
  H.263, H.264 (in 3GP or MP4 container), MPEG-4 SP, AMR, AMR-WB (in 3GP
  container), AAC, HE-AAC (in MP4 or 3GP container), MP3, MIDI, WAV, JPEG,
  PNG, GIF and BMP.
- **Additional hardware support:** Android can use video/still cameras, touch screens, GPS, accelerometers, magnetometers, and accelerated 3D graphics. Development environment: Includes a device emulator, tools for debugging, memory and performance profiling, and a plug-in for the Eclipse IDE.
- **Market:** Like many phone-based application stores, the Android Market is a catalogue of applications that can be downloaded and installed to target hardware
  over-the-air, without the use of a PC. Originally only free applications were supported. Paid-for applications have been available on the Android Market in the
  United States since 19 February 2009.The Android Market has been expanding
  rapidly.
- **Multi-touch:** Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero. The feature was initially disabled at the kernel level (possibly to avoid infringing Apple's patents on touch-

screen technology). Google has since released an update for the Nexus One and
the Motorola Droid which enables multi-touch natively.

2.1.3 SMACK API

Smack is an open source, highly modular, easy to use, XMPP client library written in Java for Java SE compatible JVMs and Android.

A pure Java library, it can be embedded into your applications to create anything from a full XMPP instant messaging client to simple XMPP integrations such as sending notification messages and presence-enabling devices. Smack and XMPP allows you to easily exchange data, in various ways e.g. fire-and-forget, publish-subscribe, between human and non-human endpoints (M2M, IoT, etc.).

Smack is a library for communicating with XMPP servers to perform real-time communications, including instant messaging and group chat.

Key Advantages of SMACK:

- Extremely simple to use, yet powerful API. Sending a text message to a user can be accomplished in only a few lines of code:

```
AbstractXMPPConnection connection =newXMPPTCPConnection("mtucker",
"password", "jabber.org");
connection.connect().login();

Messagemessage=newMessage("jsmith@jivesoftware.com", "Howdy! How are
you?");
connection.sendStanza(message);
```

- Doesn't force you to code at the XMPP protocol level, as other libraries do. Smack provides intelligent higher-level constructs such as
the Chat and Roster classes, which let you program more efficiently.
- Does not require that you're familiar with the XMPP XML format, or even that you're familiar with XML.
- Provides easy machine to machine communication. Smack lets you set any number of properties on each message, including properties that are Java objects.
- Open Source under the Apache License 2.0, which means you can incorporate Smack into your commercial or non-commercial applications.

2.1.4 OPENFIRE SERVER

Openfire (previously known as **Wildfire**, and **Jive Messenger**) is an instant messaging (IM) and groupchat server that uses XMPP server written in Java and licensed under the Apache License 2.0.

The project was originated by Jive Software in around 2002, partly in order to support their FastPath web-based customer support tool, as **Jive Messenger**, and renamed to **Wildfire** in 2005. Due to a trademark issue, it was further renamed to **Openfire** in 2007. The project was wholly handed to the community in 2008. Jive continued to host the project until 2016.

Openfire is developed under a community model, as part of the Ignite Realtime project. The project lead is Dave Cridland.

Most administration of the server is done through a web interface, which runs on the ports 9090 (HTTP) and 9091 (HTTPS) by default. Administrators can connect from anywhere and edit the server and configuration settings.

Openfire supports the following features:

- Web-based administration panel
- Plugin interface
- Customizable
- SSL/TLS support
- User-friendly web interface and guided installation
- Database connectivity (i.e. embedded HSQLDB or other DBMS with JDBC 3
- LDAP connectivity
- Platform independent, pure Java

- driver) for storing messages and user details
- Full integration with Spark (XMPP client)
- Can support more than 50,000 concurrent users

Openfire has strong support for plugins and customized builds; there are numerous plugins available for immediate download and install via the admin console, and many installations have bespoke plugins.

Openfire allows multiple server instances to work together in one clustered environment. There is an open-source clustering plugin based on open-source Hazelcast technology.

2.1.5 MICROSOFT LUIS

Language Understanding Intelligent Service (LUIS) enables developers to build smart applications that can understand human language and react accordingly to user requests. LUIS uses the power of machine learning to solve the difficult problem of extracting meaning from natural language input, so that your application doesn't have to. Any client application that converses with users, like a dialog system or a chat bot, can pass user input to a LUIS app and receive results that provide natural language understanding.

A LUIS app is a place for a developer to define a custom language model. The output of a LUIS app is a web service with an HTTP endpoint that you reference from your client application to add natural language understanding to it. A LUIS app takes a user utterance and extracts intents and entities that correspond to activities in the client application's logic. Your client application can then take appropriate action based on the user intentions that LUIS recognizes.

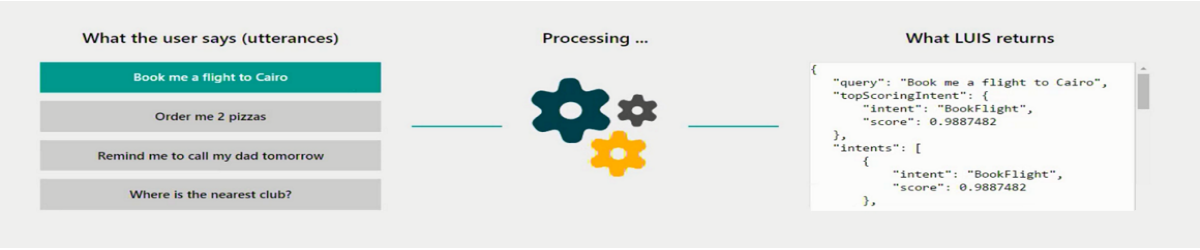

Figure 2.2: LUIS Process Overview

Key concepts in LUIS:

- **Utterance:** An utterance is the textual input from the user, that your app needs to interpret. It may be a sentence, like "Book me a ticket to Paris", or a fragment of a sentence, like "Booking" or "Paris flight." Utterances aren't always well-formed, and there can be many utterance variations for a particular intent.
- **Intents:** Intents are like verbs in a sentence. An intent represents actions the user wants to perform. It is a purpose or goal expressed in a user's input, such as booking a flight, paying a bill, or finding a news article. You define a set of named intents that correspond to actions users want to take in your application. A travel app may define an intent named "BookFlight", that LUIS extracts from the utterance "Book me a ticket to Paris".
- **Entities:** If intents are verbs, then entities are nouns. An entity represents an instance of a class of object that is relevant to a user's intent. In the utterance "Book me a ticket to Paris", "Paris" is an entity of type location. By recognizing the entities that are mentioned in the user's input, LUIS helps you choose the specific actions to take to fulfil an intent.

2.1.6 API.AI

API.AI is a natural language understanding platform that makes it easy for developers (and non-developers) to design and integrate intelligent and sophisticated conversational user interfaces into mobile apps, web applications, devices, and bots.

The diagram below shows how API.AI is related to other components and how it processes data:

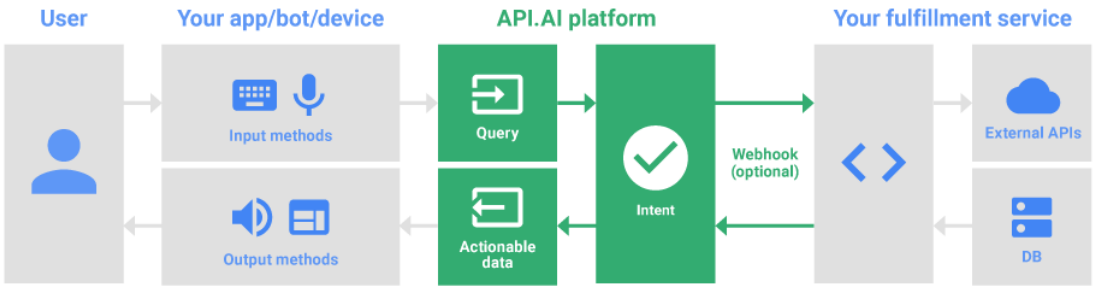

Figure 2.3: API.AI Overview

In the diagram, the green is provided by the API.AI platform. Your app / bot / device code provides the input and output methods and responds to actionable data. You can also provide an optional webhook implementation which API.AI uses to connect to your web service. Your web service can then perform business logic, call external APIs, or access data stores.

API.AI receives a **query** as input data. A query is either text in natural language or an event name sent to API.AI. API.AI matches the query to the most suitable **intent** based on information contained in the intent (examples, entities used for annotations, contexts, parameters, events) and the agent's machine learning model. API.AI transforms the query text into **actionable data** and returns output data as a JSON response object.

The process of transforming natural language into actionable data is called Natural Language Understanding (NLU). Dialog management tools such as contexts and intent priorities allow developers to control the conversation flow.

2.1.7 MICROSOFT TRANSLATOR TEXT API

Microsoft Translator Text API, part of the Microsoft Cognitive Services API collection, is a cloud-based machine translation service supporting multiple languages that reach more than 95% of world's gross domestic product (GDP). Translator can be used to build applications, websites, tools, or any solution requiring multi-language support.

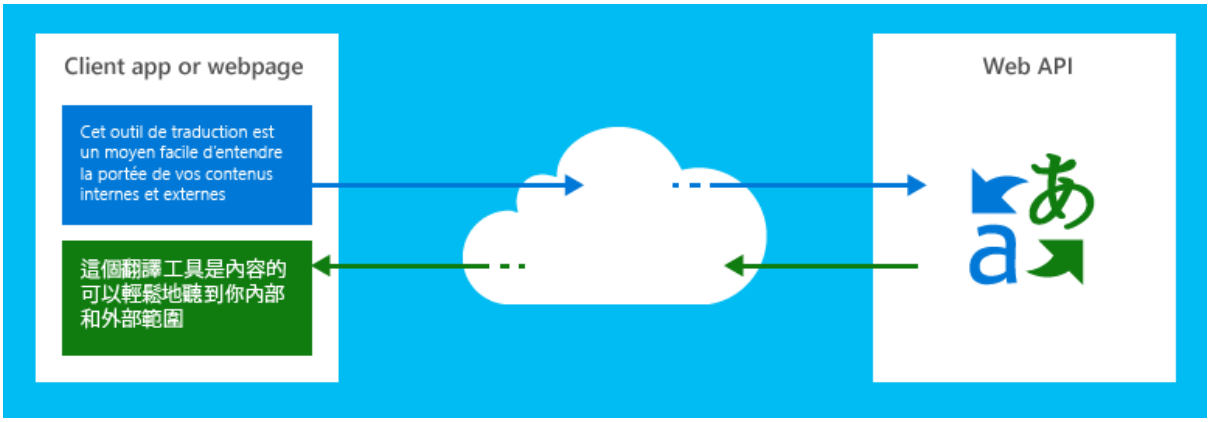

Figure 2.4: Microsoft Translator

Microsoft Translator text API has been used by Microsoft groups since 2007 and is

available as an API for customers since 2011. The Microsoft Translator text API is used extensively within Microsoft. It is incorporated across product localization, support, and online communication teams (e.g., Windows blog). This same service is also accessible, at no additional cost, from within familiar Microsoft products such as Bing, Cortana, Microsoft Edge, Office, SharePoint, Skype, and Yammer.
Microsoft Translator can be used in web or client applications on any hardware platform and with any operating system to perform language translation and other language-related operations such as language detection, text to speech, or dictionary.

Leveraging industry standard REST technology, the developer sends source text (or audio for the speech API) to the service with a parameter indicating the target language, and the service sends back the translated text for the client or web app to use.

The Microsoft Translator service is an Azure service hosted in Microsoft data centres and benefits from the security, scalability, reliability, and nonstop availability that other Microsoft cloud services also receive.

2.1.8 APACHE

The name 'Apache' was chosen with respect for the Native American Indian tribe of Apache, well-known for their superior skills in warfare strategy and their inexhaustible endurance. It also makes a cute pun on "a patchy web server" -- a server made from a series of patches -- but this was not its origin. The group of developers who released this new software soon started to call themselves the "Apache Group".

Apache is a freely available Web server that is distributed under an "open source" license. Version 2.0 runs on most UNIX-based operating systems (such as Linux, Solaris, Digital UNIX, and AIX), on other UNIX/POSIX-derived systems (such as Rhapsody, BeOS, and BS2000/OSD), on AmigaOS, and on Windows 2000. According to a Netcraft Web server survey, 60% of all Web sites on the Internet are using Apache (62% including Apache derivatives), making Apache more widely used than all other Web servers combined.

2.2 HARDWARE REQUIREMENTS

2.2.1 SMARTPHONE

A smartphone is a mobile personal computer with an advanced mobile operating system with features useful for mobile or handheld use. Smartphones, which are typically pocket-sized (as opposed to tablets, which are much larger in measurement), have the ability to place and receive voice/video calls and create and receive text messages, have personal digital assistants (such as Siri, Google Assistant, Alexa, or Cortana), an event calendar, a media player, video games, GPS navigation, digital camera and digital video camera. Smartphones can access the Internet through cellular or Wi-Fi and can run a variety of third-party software components ("apps" from places like Google Play Store or Apple App Store). They typically have a colour display with a graphical user interface that covers more than 76% of the front surface. The display is almost always a touchscreen and sometimes additionally a touch-enabled keyboard like the Priv/Passport BlackBerrys, which enables the user to use a virtual keyboard to type words and numbers and press onscreen icons to activate "app" features.

Figure 2.5: Android Smartphone

In 1999, the Japanese firm NTT DoCoMo released the first smartphones to achieve mass adoption within a country. Smartphones became widespread in the late 2000s. Most of those produced from 2012 onward have high-speed mobile broadband 4G LTE, motion sensors, and mobile payment features. In the third quarter of 2012, one billion smartphones were in use worldwide. Global smartphone sales surpassed the sales figures for regular cell phones in early 2013.

2.3 NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements are requirements which impose constraints on the design or implementation such as performance engineering requirements, quality standards or design constraints. It is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours.

2.3.1 USABILITY

Usability is the ease of use and learnability of a human-made object. In software engineering, usability is the degree to which a software can be used by specified consumers to achieve quantified objectives with effectiveness, efficiency, and satisfaction in a quantified context of use.

2.3.2 RELIABILITY

Reliability is an attribute of any computer-related component (software, or hardware, or a network, for example) that consistently performs according to its specifications. It has long been considered one of three related attributes that must be considered when making, buying, or using a computer product or component.

2.3.3 PERFORMANCE

Computer performance is the amount of work accomplished by a computer system. Depending on the context, high computer performance may involve one or more of the following:

- Short response time for a given piece of work

- High throughput (rate of processing work)
- Low utilization of computing resource(s)
- High availability of the computing system or application
- Fast (or highly compact) data compression and decompression
- High bandwidth
- Short data transmission time.

2.3.4 SCALABILITY

Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged in order to accommodate that growth. For example, it can refer to the capability of a system to increase its total output under an increased load when resources (typically hardware) are added. An analogous meaning is implied when the word is used in an economic context, where scalability of a company implies that the underlying business model offers the potential for economic growth within the company.

2.3.5 MAINTAINABILITY

Maintainability is a measure of the ease and rapidity with which a system or equipment can be restored to operational status following a failure.

2.3.6 PORTABILITY

The system should be easily portable to another system. This is required when the web server, which is hosting the system, gets stuck due to some problems, which requires the system to be taken to another system.

2.3.7 REUSABILITY

The system should be divided into such modules that it could be used as a part of another system without requiring much of work.

2.3.8 FLEXIBILITY

The system should be flexible enough to allow modifications at any point of time.

DESIGN

CHAPTER 3

DESIGN

3.1 OVERVIEW

Design is the second stage in the software life cycle of any engineered product or system. Design is a creative process. A good design is the key to effective system. It may be defined as "the process of applying various techniques and principles for the purpose of defining a process or a system in sufficient detail to permit its physical realization."

The design specification describes the features of the system, the components or elements of the system and their appearance to the end users. In system design, high-end decisions are taken regarding the basic system architectures, platforms and tools to be used. The system design transforms a logical representation of what a given system is required into the physical specification. The significant design factors to be considered are reliability, response time, throughput of the system, maintainability, expandability etc.

Design is the place where quality is fostered in software development. During design, decisions are made that ultimately affect the success of the software construction. Software design is an iterative process through which the requirements are translated into a "blue print" for constructing the software. Software design serves as the foundation for all software engineering and software maintenance steps that follow.

3.2 SYSTEM ARCHITECTURE

The proposed system architecture contains three main modules: Instant Messaging (android client and Openfire server), Reminder, and Summarizer modules. The Instant Messaging client is further divided into three sub modules: Reminder Alarm List, Messaging, and Summary List (**Figure 3.1**).

The working of our model is as follows.

The user logs in to the IMA application (Instant Messaging client) using his username and password. If he/she does not have one, he/she can create an account. The user is then met

with the chat selection screen; here, he/she can select the recipient with whom he would like to communicate.

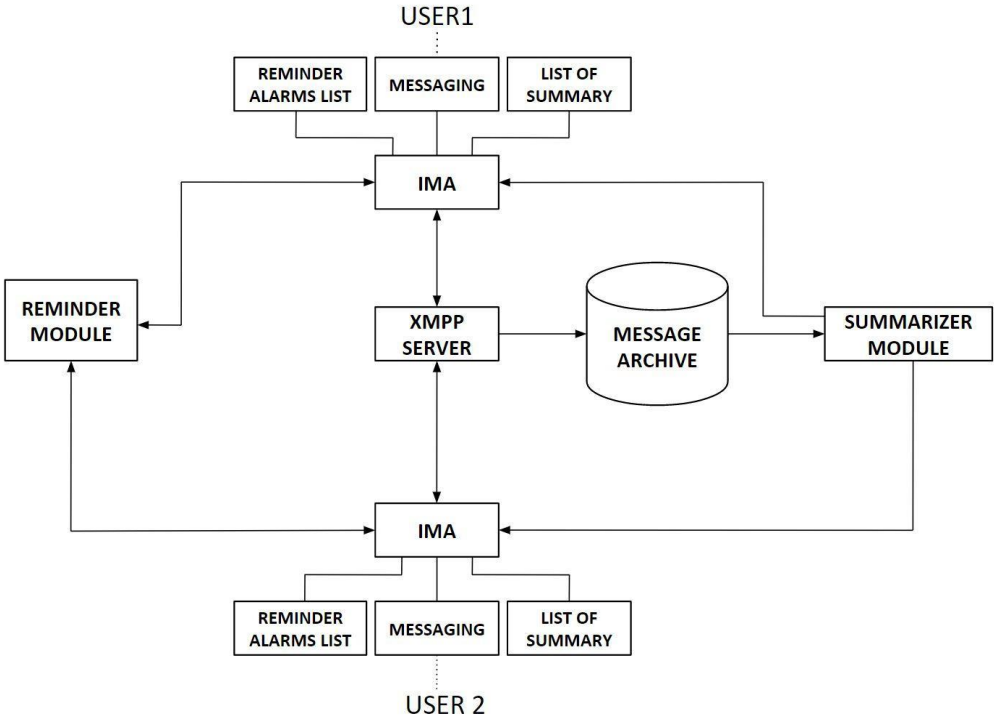

Figure 3.1: System Architecture

The XMPP server (Instant Messaging Server) transfers message between users. and also archives the message in Message Archive database whenever IMA receives the message the Reminder Alarm List sub module of IMA send it to Reminder module. The Reminder module are actually API calls to ***Luis (by Microsoft)*** or to ***API.AI*** (Google) which are language understanding systems. These APIs are initially trained using a training data set to be on the look-out for words or phrases that might be a viable candidate for a reminder. The APIs implement **machine learning algorithms**, and hence, are capable of learning ways in which a word or a phrase can be classified as a reminder candidate, (say, by looking-out for synonyms of the previously defined words from the training data set). By doing so, the API becomes increasingly accustomed to the user's style of conversation and will be better able to determine potential reminders in the future. When a sentence has been classified as a candidate for a reminder, it is broken down in order to determine the various parameters of

the reminder, such as the subject, the object, the action, the location, date-time parameters, and so on. Once the parameters have been determined, they are sent back to the IMA application, where the Reminder Alarm List module set these reminders as alarms and is also made available for the user as a list. Meanwhile, the user can ask for summary of his conversation by clicking a button on respective Contact List in messaging module, which triggers the Summary List sub module of IMA. The Summary List sub module makes request for remote Summarizer module for summary. The Summarizer module retrieves the conversation from message archive and apply summarizer algorithm as described in the previous section. The result is sent back to the IMA application where it is displayed to the user.

3.3 USE CASE DIAGRAM

A use case diagram is a graphic depiction of the interactions among the elements of a system. It is a methodology used in system analysis to identify, clarify, and organize system requirements. In this context, the term "system" refers to something being developed or operated. Use case diagrams are employed in UML (Unified Modelling Language), a standard notation for the modelling of real-world objects and systems.

System objectives can include planning overall requirements, validating a hardware design, testing and debugging a software product under development, creating an online help reference, or performing a consumer-service-oriented task. For example, use cases in a product sales environment would include item ordering, catalogue updating, payment processing, and customer relations. A use case diagram contains four components, which are described as follows:

- The boundary, which defines the system of interest in relation to the world around it.
- The actors, usually individuals involved with the system defined according to their roles.
- The use cases, which are the specific roles played by the actors within and around the system.
- The relationships between and among the actors and the use cases.

The use case diagram for this project is given below in **Figure 3.2**.

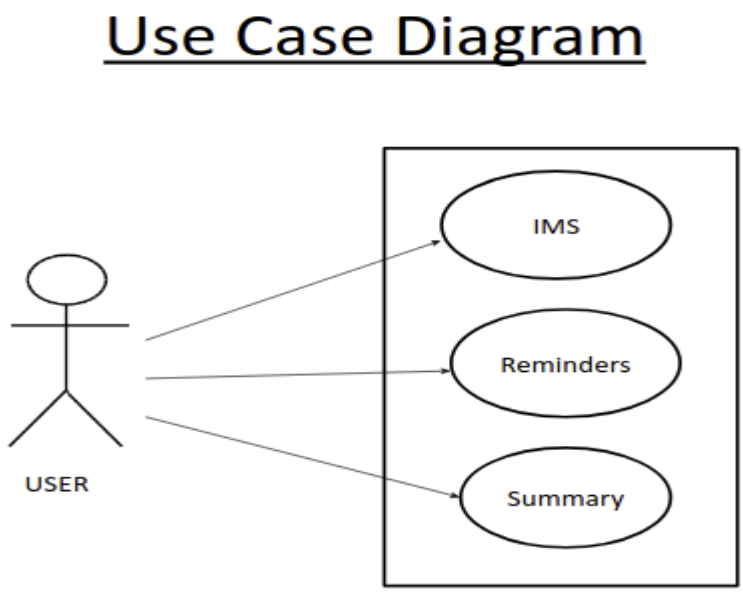

Figure 3.2: Use Case Diagram

- IMS: Is the Android mobile application where user will send and receive the messages from and to a particular user.
- Reminders: The conversation or the chat will be analysed and if any imperative or ordering statement occurs then accordingly reminder will be set and will intimate the user accordingly.
- Summary: If a particular user requires a summary of the chat from a particular person he will request for the summary and will be displayed to him accordingly.

3.4 ACTIVITY DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, activity diagrams are intended to model both computational and organizational processes (i.e. workflows). Activity diagrams show the overall flow of control.

Activity diagrams are constructed from a limited number of shapes, connected with arrows. The most important shape types:

- Rounded rectangles represent actions;
- Diamonds represent decisions;

- Bars represent the start (split) or end (join) of concurrent activities;
- A black circle represents the start (initial state) of the workflow;
- An encircled black circle represents the end (final state).

Arrows run from the start towards the end and represent the order in which activities happen. Activity diagrams may be regarded as a form of flowchart. Typical flowchart techniques lack constructs for expressing concurrency. However, the join and split symbols in activity diagrams only resolve this for simple cases; the meaning of the model is not clear when they are arbitrarily combined with decisions or loops.

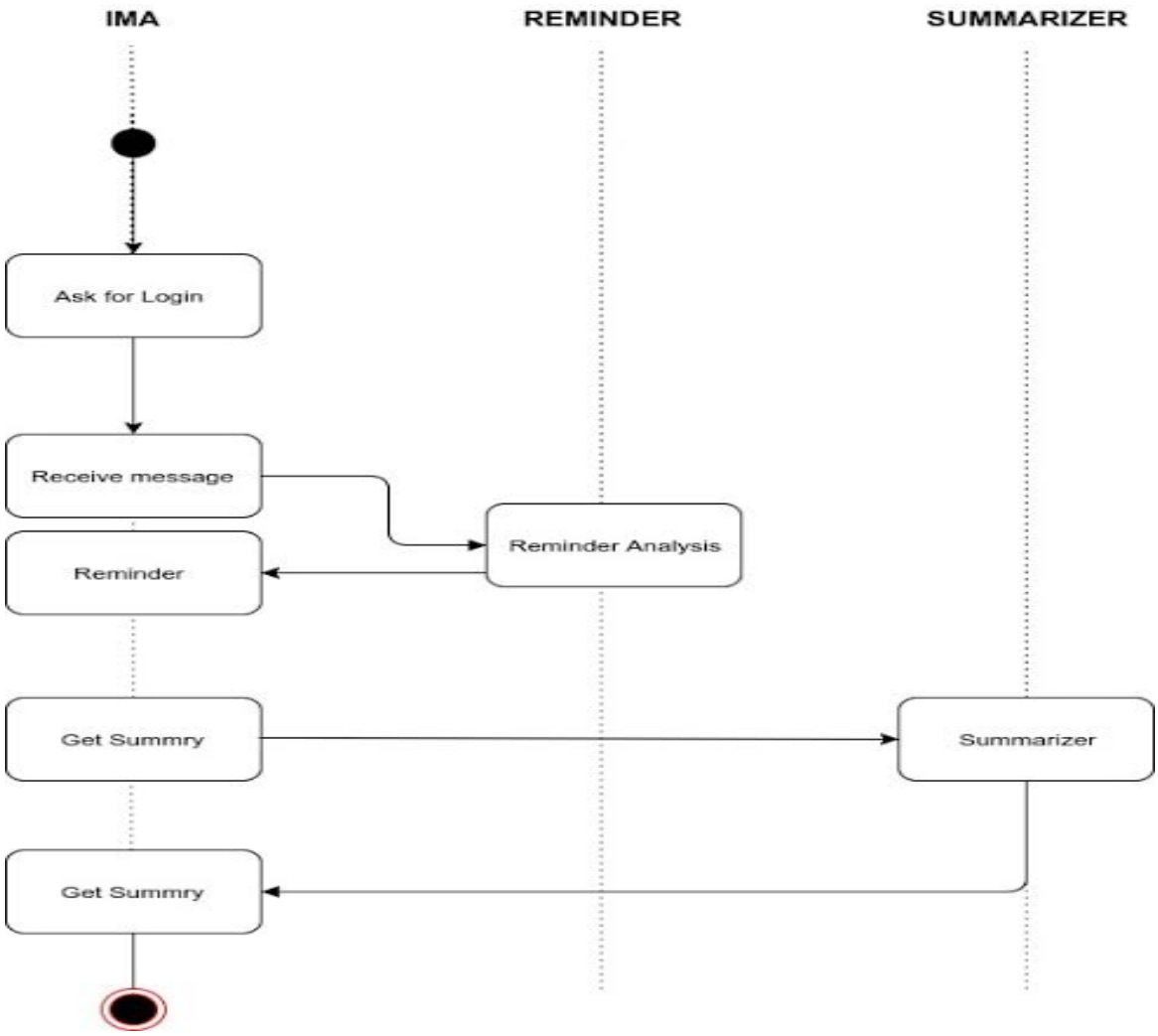

Figure 3.3: Activity Diagram

The user sends a message to a particular user through IMS. IMS in turn will be in connection with intelligent end-points and all the messages will be analysed through it at any

given point of time. If any imperative sentences or eventful statements occur, then such sentences are recognized and the reminder will be set for the user based on the sentence. If the user requests for the summary of the conversation, it analyses the entire conversation, removes all the unnecessary phrases, and provides the user with a brief summary of that particular conversation.

3.5 DATA FLOW DIAGRAM

A **data flow diagram** (**DFD**) is a graphical representation of the "flow" of data through an information system, modelling its process aspects. A DFD is often used as a preliminary step to create an overview of the system, which can later be elaborated. DFDs can also be used for the visualization of data processing (structured design).

A DFD shows what kind of information will be input to and output from the system, where the data will come from and go to, and where the data will be stored. It does not show information about the timing of process or information about whether processes will operate in sequence or in parallel (which is shown on a flow chart).

DFDs are constructed using four major components:

- External entries
- Data stores
- Processes
- Data Flows

The DFD for this project is described in **Figure 3.4**. The user uses the messaging client, IMS, to chat with a contact listed in his/her contact list. These messages will be sent to intelligent text analyser. Here, the entire chat history will be taken and the sentences in the chat will be analysed. If any imperative sentences or eventful statements occur, then such sentences are recognized and sent to the reminder module. The reminder will be set for the user based on the sentence. If the user requests for the summary of the conversation, then the summary module analyses the conversation, removes unnecessary words and phrases, and delivers the summary to the user.

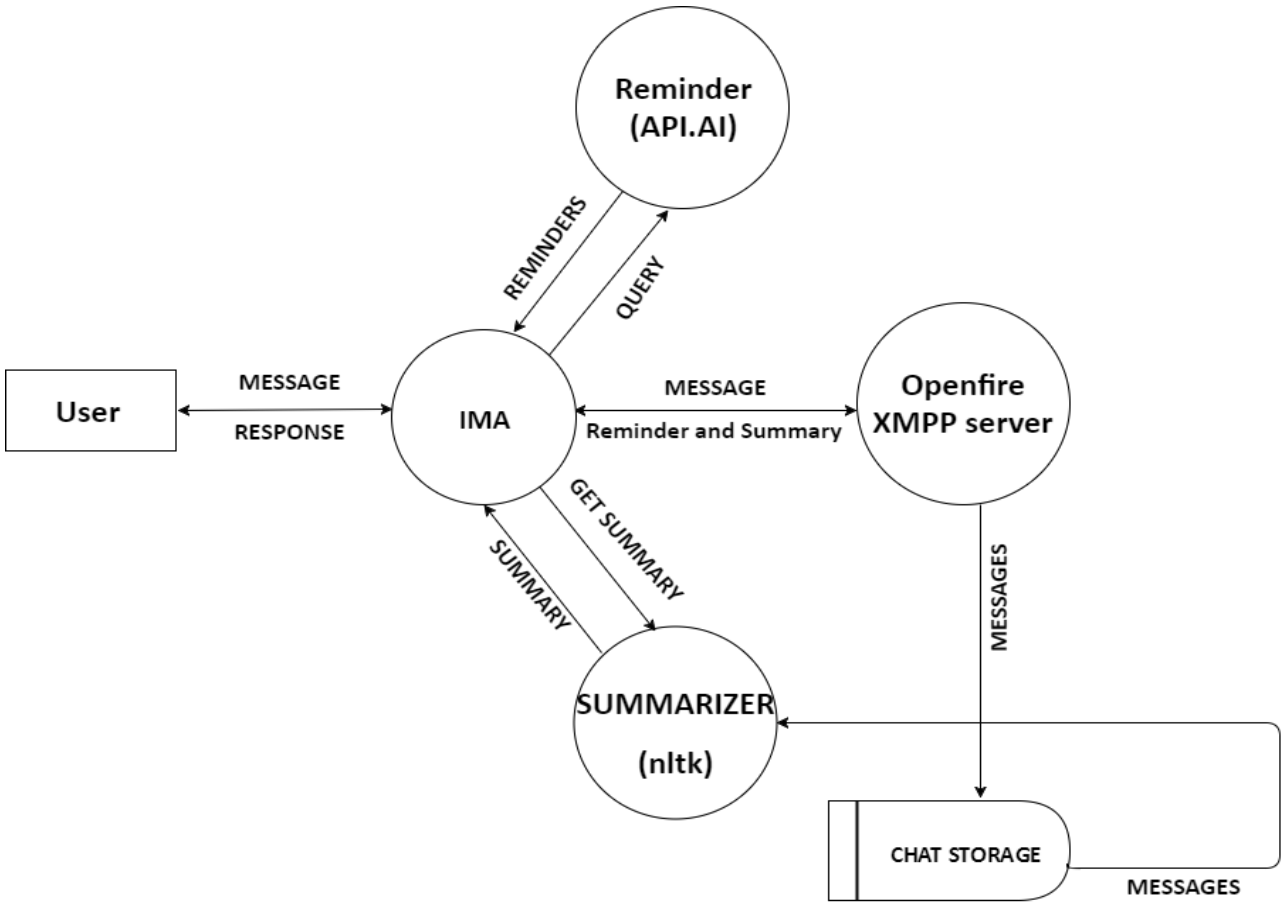

Figure 3.4: Data Flow Diagram

3.6 FUNCTIONALITY

Most of the existing Instant Messaging applications do not give the option of a personal conversation via an API call, as it is matter of privacy. Even if they do provide, it is not flexible for analysis of plausible reminders and summarization of conversation in real time. Thus, we have to develop our own Instant Messaging application prototype which we have called Intelligent Messaging Assistant (IMA). The IMA should provide three main functionalities for users:

- Instant Messaging
- Reminder Setting
- Conversation Summarization

3.6.1 Instant Messaging

The app will provide one-to-one and group conversation. We restrict the conversation to be only textual-based for this project. We use a widely adopted open protocol for instant messaging, namely, Extensible Messaging and Presence Protocol (XMPP). XMPP is a communication protocol for message-oriented middleware based on XML (Extensible Mark-up Language). It enables near real-time exchange of structured, yet extensible data between two or more network entities. Since XMPP is an open standard, many implementations of the server, client, and the library are available. Therefore, we use Openfire as the XMPP server for instant messaging. Openfire is a real-time collaboration (RTC) server licensed under the Open Source Apache License. The IMA is developed using the spark library.

3.6.2 Reminder Setting

In the scope of this project, eventful sentences are those which give or request some task or responsibility to one of the participants of the conversation. We use machine learning binary classification algorithm for finding whether the given sentence qualifies as a candidate for an eventful sentence. The input to this algorithm is a message and the output is whether the given sentence is a reminder sentence (positive classification) or not (negative classification). The algorithm for determining reminders is described as follows:

- A conversation often contains introductory phrases such as “hey”, “hello”, “what’s up?”, “how are you”, and so on. These messages are not considered for plausible reminder analysis.
- Stop words such as “ah”, “oh”, “sup” etc. are removed from the message.
- It is known from experience that the eventful sentences will have at least 10 characters in it. Thus, any message containing lesser than 10 characters are not considered for plausible remainder analysis.
- The sentences that pass all of the above condition steps are considered for plausible remainder analysis and is fed to classifier algorithm.

- If the output of the classifier is positive i.e., the sentence is a eventful sentence, it is set as a reminder.

The degree of correctness in classification achieved by the algorithm depends on feature selection. Feature selection therefore plays an important role part in this project. The following are different features useful in sentence classification:

- **Predefined Phrases and Keywords:** Sentences containing words, phrases, or keyword such as 'remind me', 'remember', 'ping', 'notify', 'get' etc. are strong candidates for being classified as a reminder. One of the features checks for the existence of such words.
- **Adverb-Verb Dependency or Imperative Sentence:** Since many imperative sentences will have an adverb followed by a verb, this feature is the most useful for our classifier algorithm. For example, in the sentence please bring pen drive tomorrow. Here please is the adverb and bring is the verb.
- **Existence of location and time with a verb in sentence:** Messages that have time, date, location, or a combination of any of these three characteristics are likely to be candidates for a reminder.

3.6.3 Conversation Summarization

Summarization has become an important and timely tool for assisting and interpreting information in today's fast-growing information age. Summarization focuses on building extractive, generic, and surface-level-feature-based summarizers. These extractive summarizers select and present pieces of the original speech transcripts or audio segments as summaries, rather than rephrase or rewrite them. The output summary could be textual (transcripts) or spoken (e.g., concatenated audio clips). This project will summarize the conversations which often contain filler words. Thus, removing these filler words in the transcripts is the first step in summarizer. The pieces to be extracted from the transcripts can be done by assigning weights to each sentence based on certain features and methodologies. The features that can be considered for extractive summarization are as follows:

- **Content Word (Keyword) Feature:** Content words or Keywords are usually nouns

and determined using tf × idf measure. Sentences having keywords are of greater chances to be included in the summary.

- **Sentence Length Feature:** Very short sentences are not included in the summary.
- **Cue-Phrase Feature:** Sentences containing any cue phrases (e.g. “in conclusion”, "in short", “this report”, “summary”, “argue”, “purpose”, “develop”, “attempt” etc.) are most likely to be in the summary.

The summarization algorithm uses basic summarization features and builds from it. Those features are: title feature, sentences length, sentence position, and keyword frequency.

- Title feature is used to score the sentence with the regards to the title.
- Sentence length is scored depends on how many words are in the sentence.
- Sentence position is where the sentence is located. The introduction and conclusion will have higher score for this feature.
- Keyword frequency is just the frequency of the words used in the whole text.

The algorithm works as follows:

- The entire conversation is fed to the algorithm in the form of a file (a single file is more convenient to handle longer conversations or large bodies of text). The features are determined by the summarizer function by implementing the **parser module.** This module uses the natural language toolkit (NLTK), which works towards analysing and interpreting the languages humans use naturally.
- The NLTK functions enables the parser to separate every sentence in the summary file, split individual words from each sentence, remove punctuations (as they are not necessary), determine and remove stop words (present at the end of typical sentences), determine keywords, and send these to the **summarizer function**.
- The **summarize function** handles the summarization of the algorithm. The title and the input file is fed into the function as input parameters. The summarizer function determines the title feature by finding out the sentence score, sentence length, sentence position and keyword frequency. The function appends these parameters which is sent as the final summary.
- The final summary is then compiled and is ready to be presented to the user on his/her request.

IMPLEMENTATION

CHAPTER 4

IMPLEMENTATION

A **software implementation** method is a systematically structured approach to effectively integrate a software based service or component into the workflow of an individual end-user.

The project has three main modules:
Instant Messaging, Reminder, and Conversation Summarizer.

4.1 Instant Messaging

Instant messaging module is subdivided into the XMPP server module and the XMPP client. The XMPP server is based on Openfire server, which is the best available open XMPP server written in java in the market.
This enables us to maintain a custom database to archive messages, and has many plug-ins to create a user account, retrieve messages, and so on.

Code to create users using REST API plugin at server by making restful call Openfire server using Retrofit 2 android library.

```
mApiService.createUser("jXd63gwtrrqgX3J7","application/json",userOpenfire).enqueue(
new Callback<Void>() {
 @Override
 public void onResponse(Call<Void> call, Response<Void> response) {

  //user created
   Log.d(TAG,response.toString());
   onSignupSuccess();

 }
 @Override
 public void onFailure(Call<Void> call, Throwable t) {
```

```
    //something was wrong while creating server
     onSignupFailed();
     Log.d(TAG,"SIGNUp FAILED "+t.toString());
     progressDialog.setMessage("failed to signup");
  }
});
```

The Xmpp client is our instant messaging android application called Intelligent Messaging Assistant (IMA). The IMA application has three sub modules namely Messaging, Reminder/Alarm setting, and summary.

The Messaging module performs the following functions:

- Login or SignUp to server
- Retrieve user contacts and present the contact list
- Provide chat functionality

Login to Server:

- Get credentials from the user.
- Establish a connection to the server using xmpp protocol

```
        XMPPTCPConnectionConfiguration.Builder
    builder=XMPPTCPConnectionConfiguration.builder();
         builder.setServiceName(SERVICE_ADDRESS);
        builder.setHost(HOST_ADDRESS);
        builder.setDebuggerEnabled(true);
        builder.setPort(5222);
        builder.setUsernameAndPassword(username,password);


        builder.setSecurityMode(ConnectionConfiguration.SecurityMode.disabled);
        Log.d(TAG,"trying to login");
```

```
xmpptcpConnection=new XMPPTCPConnection(builder.build());
xmpptcpConnection.addConnectionListener(this);
xmpptcpConnection.connect();
```

- Login to his/her account, and on success, retrieve the user's contact list.

Retrieve user contacts and present the contact list:

- Retrieve user contacts

```
Roster roster=Roster.getInstanceFor(xmpptcpConnection);
if(!roster.isLoaded()) {
try {
roster.reloadAndWait();
} catch (SmackException.NotLoggedInException e) {
e.printStackTrace();
} catch (SmackException.NotConnectedException e) {
e.printStackTrace();
}
}
roster.addRosterListener(this);
Collection<RosterEntry> entries=roster.getEntries();
roster.setSubscriptionMode(Roster.SubscriptionMode.accept_all);
```

- Display contact list to user

- Add contacts to ContactList model

```
public void addUser(User user) {
list.add(user);
ContentValues cv=new ContentValues();
cv.put(DataBaseMetaData.TABLE_USERS_COLUMN2,user.getUserName());
```

```
        cv.put(DataBaseMetaData.TABLE_USERS_COLUMN3,String.valueOf(System.currentTimeMillis()));
context.getContentResolver().insert(Uri.parse(DataBaseMetaData.TABLE_USER_URI_STRING),cv);
        }
```

Provide Chat Functionality:

- Listen to message from other users

```
if(message.getType()== Message.Type.chat&&message.getBody()!=null) {
contactList.addMessage(message.getBody(),System.currentTimeMillis(),fromJid.split("@")[0],ChatMessage.Type.RECEIVED);
```

- Notify the UI to update messages in display

```
Intent intent=new Intent(ACTION_RECEIVED_MESSAGE);
        String fromJid=chat.getParticipant();
        String msg=message.getBody();
        context.sendBroadcast(intent);
```

- Send messages to the other user

```
    Chat newChat=ChatManager.getInstanceFor(connection.xmpptcpConnection).createChat(toJID;
        Message messageObj=new Message(toJID);
        messageObj.setBody(message);
        messageObj.setType(Message.Type.chat);
        try {
        newChat.sendMessage(message);
        }
         catch (SmackException.NotConnectedException e) {
```

```
e.printStackTrace();
}
```

The Reminder Setter sub-module of IMA performs the following:

- Check the length of the message
- If the length of the message is greater than 10, send it to **Reminder Main module** for further analysis to know if the message is an eventful sentence. Else, it does nothing.

```
if(message.getBody().length()>8) {
 ReminderList.getInstance(context).callAPiAi(mymsg,fromJid.split("@")[0]);
```

- If the message is an eventful sentence, then set the reminder

```
if(responseObject!=null)
if(responseObject.getResult()!=null)
if(responseObject.getResult().getParameters()!=null)
if(response.body().getResult().getParameters().getObject()!=null&response.body().getResult(
).getScore()>0.65)
setReminder(responseObject, user)
```

The **Summary** sub-modules ends a request to our apache server which is hosting the Main Summarizer module and presents the summary response to user.

```
mApiService.getSummary(map).enqueue(new Callback<SummaryModel>() {
  @Overrideis
  public void onResponse(Call<SummaryModel> call, Response<SummaryModel>
response) {
     summaryModel=response.body();
     String summary="";
     for(String s:summaryModel.getSentList())
```

```
        summary+=s+" \n";

SummaryText.setText(summary);
    Log.d(TAG,summaryModel.toString());

  }
```

4.2 Reminder Module

Our reminder module is an API.AI agent. This agent runs a machine learning algorithm to classify sentences as candidate for reminder, i.e., whether it is an eventful sentence or not. Labels help the classifier algorithm to classify the conversation message into eventful sentences or not. The probability of the classification depends on the number of labels annotable to the sentence. We have considered following labels to annotate the training samples.

- @Reminders: This is a label used to annotate predefined keyword such as “get”, “bring”, “remind” ,“meet me” and few more.
- @Location: This is a label used to annotate common location such as home, office, cabin, canteen etc.
- @Object: This is a label used to annotate objects in sentence like book, file, pendrive etc.
- @recurrence: This defines days of the week, or months of the year, which occur on a recurring basis in a conversation.
- @sys.date: An API.AI inbuilt label that recognises date information.
- @sys.time: An API.AI inbuilt label that recognises time information.

Initially, we had labelled 30 sentences manually for training the classifier. The training set is regularly updated with new examples and classifier is retrained. Given below are the figures that depict this training process. **Figure 4.1** shows the labelled training sentences and **Figure 4.2** shows the labels being annotated to the training examples.

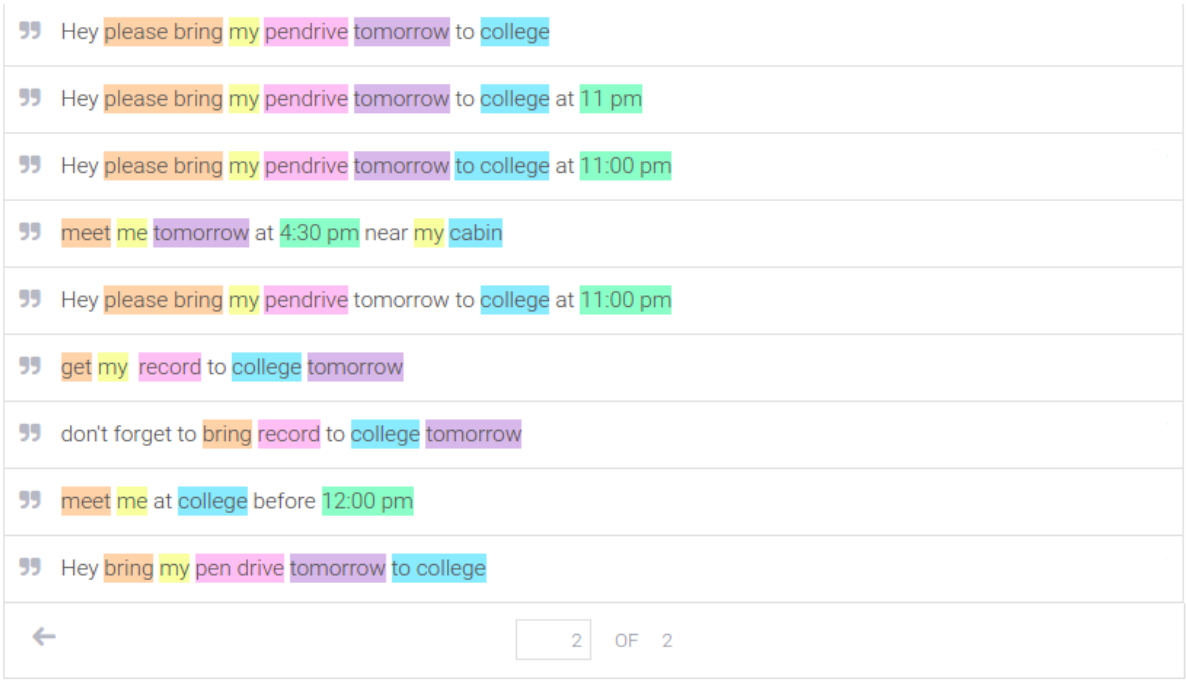

Figure 4.1: API.AI Training Samples

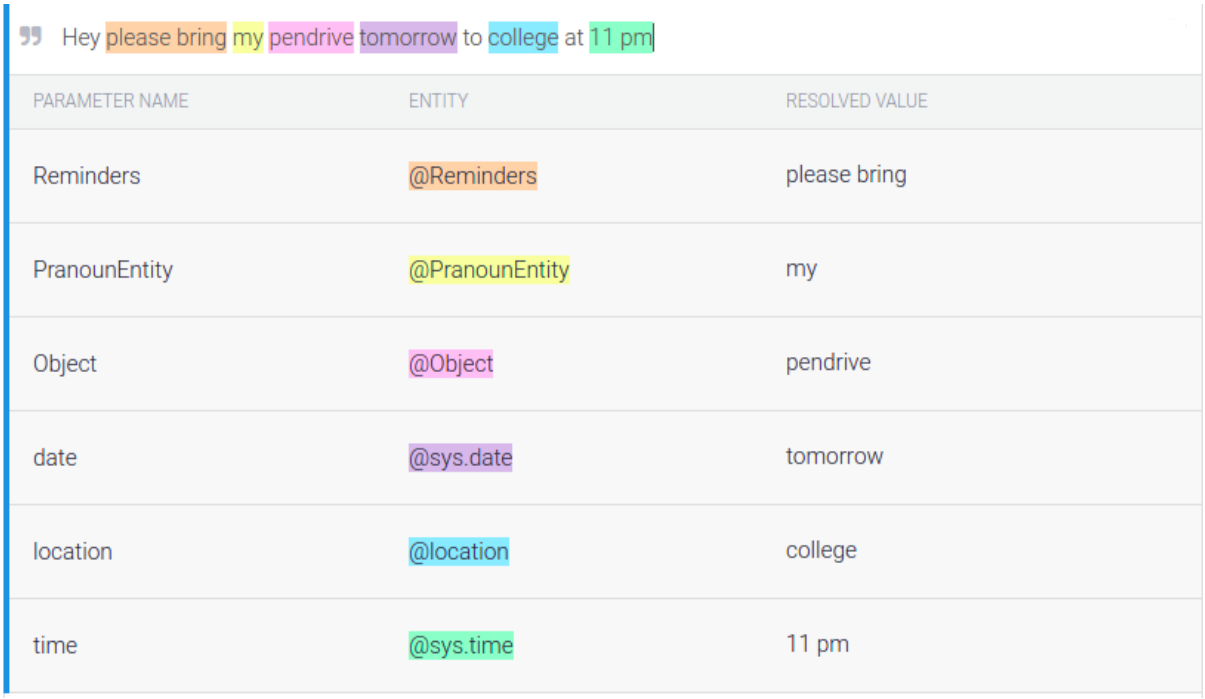

Figure 4.2: API.AI Annotated Training Example 1

PARAMETER NAME	ENTITY	RESOLVED VALUE
Reminders	@Reminders	get
PranounEntity	@PranounEntity	my
Object	@Object	record
location	@location	college
date	@sys.date	tomorrow

Figure 4.3: API.AI Annotated Training Example 2

When the API.AI agent receives a request from client, i.e., from IMA android application, it feeds the message to the classification algorithm, which analyses the message and classifies it as either an eventful sentence or not. The classification result is returned back to IMA as a json response. The sample json response contains the message being sent by the IMA, which is the statement telling whether the message is eventful or not. If the message is eventful, then the labels being assigned to it is as shown below.


```
{
 "id": "d1e82e98-c0cb-43fc-a3e4-826ca8b8bfec",
 "timestamp": "2017-05-24T05:31:26.964Z",
 "lang": "en",
 "result": {
  "source": "agent",
  "resolvedQuery": "Hey get my pendrive tomorrow to college before 12pm",
  "action": "setReminder",
  "actionIncomplete": false,
  "parameters": {
   "date": "2017-05-25",
   "location": "college",
   "Object": "pendrive",
   "PranounEntity": "my",
```

```
    "Reminders": "get",
    "time": "12:00:00"
   },
   "contexts": [],
   "metadata": {
    "intentId": "4560efb0-0e4c-42dd-b7a1-cd3bcf88d2cf",
    "webhookUsed": "false",
    "webhookForSlotFillingUsed": "false",
    "intentName": "Reminder"
   },
   "fulfillment": {
    "speech": "",
    "messages": [
     {
      "type": 0,
      "speech": ""
     }
    ]
   },
   "score": 0.97
  },
  "status": {
   "code": 200,
   "errorType": "success"
  },
  "sessionId": "343a9eb8-ce8f-4cb9-8e2d-0d126d26cd0d"
}
```


4.3 Summarizer Module

The summarizer module is responsible for creating a summary on the entire conversation to the user. This summary is a brief, but detailed description of the entire conversation, regardless of the length of the conversation. The following codes describe the various

portions that comprise this module:

Main module to obtain input and provide summary:

```
input = getInput()
input['text'] = input['text'].decode("ascii", "ignore")
input['text'] = " ".join(input['text'].replace("\n", " ").split())
summarizer = Summarizer()
result = summarizer.summarize(input['text'], input['title'], 'Undefined', 'Undefined')
result = summarizer.sortScore(result)
result = summarizer.sortSentences(result[:30])
```

Code that performs summarization:

```
from parser import Parser

class Summarizer:
    def __init__(self):
        self.parser = Parser()

    def summarize(self, text, title, source, category):
        sentences = self.parser.splitSentences(text)
        titleWords = self.parser.removePunctations(title)
        titleWords = self.parser.splitWords(title)
        (keywords, wordCount) = self.parser.getKeywords(text)

        topKeywords = self.getTopKeywords(keywords[:10], wordCount, source,
category)

        result = self.computeScore(sentences, titleWords, topKeywords)
        result = self.sortScore(result)

        return result
```

```
    def getTopKeywords(self, keywords, wordCount, source, category):
        # Add getting top keywords in the database here
        for keyword in keywords:
            articleScore = 1.0 * keyword['count'] / wordCount
            keyword['totalScore'] = articleScore * 1.5

        return keywords

    def sortScore(self, dictList):
        return sorted(dictList, key=lambda x: -x['totalScore'])

    def sortSentences(self, dictList):

        return sorted(dictList, key=lambda x: x['order'])

    def computeScore(self, sentences, titleWords, topKeywords):
        keywordList = [keyword['word'] for keyword in topKeywords]
        summaries = []

        for i, sentence in enumerate(sentences):
            sent = self.parser.removePunctations(sentence)
            words = self.parser.splitWords(sent)

            sbsFeature = self.sbs(words, topKeywords, keywordList)
            dbsFeature = self.dbs(words, topKeywords, keywordList)

            titleFeature = self.parser.getTitleScore(titleWords, words)
            sentenceLength = self.parser.getSentenceLengthScore(words)
            sentencePosition = self.parser.getSentencePositionScore(i, len(sentences))
            keywordFrequency = (sbsFeature + dbsFeature) / 2.0 * 10.0
            totalScore = (titleFeature * 1.5 + keywordFrequency * 2.0 + sentenceLength *
0.5 + sentencePosition * 1.0) / 4.0
```

```
            summaries.append({
                # 'titleFeature': titleFeature,
                # 'sentenceLength': sentenceLength,
                # 'sentencePosition': sentencePosition,
                # 'keywordFrequency': keywordFrequency,
                'totalScore': totalScore,
                'sentence': sentence,
                'order': i
            })

        return summaries

    def sbs(self, words, topKeywords, keywordList):
        score = 0.0

        if len(words) == 0:
            return 0

        for word in words:
            word = word.lower()
            index = -1

        if word in keywordList:
            index = keywordList.index(word)

        if index > -1:
            score += topKeywords[index]['totalScore']

        return 1.0 / abs(len(words)) * score

    def dbs(self, words, topKeywords, keywordList):
        k = len(list(set(words) & set(keywordList))) + 1
```

```
        summ = 0.0
        firstWord = {}
        secondWord = {}

        for i, word in enumerate(words):
            if word in keywordList:
                index = keywordList.index(word)

                if firstWord == {}:
                    firstWord = {'i': i, 'score': topKeywords[index]['totalScore']}
                else:
                    secondWord = firstWord
                    firstWord = {'i': i, 'score': topKeywords[index]['totalScore']}
                    distance = firstWord['i'] - secondWord['i']

                    summ += (firstWord['score'] * secondWord['score']) / (distance ** 2)

        return (1.0 / k * (k + 1.0)) * summ
```

This summarization module also uses additional sub-modules to determine key features necessary for providing an accurate summary, such as Title Score, Sentence Length, Sentence position, punctuation removal, keyword frequency, etc. Some of these modules are also derived from the Parser main module, explained as follows.

Code that performs token parsing for summarization:

The parser module utilizes functions from the NLTK (Natural Language Tool Kit), a to obtain the sub-modules used in the summarization process.

```
import nltk.data
import os

class Parser:
```

```
    def __init__(self):
        self.ideal = 20.0
        self.stopWords = self.getStopWords()

    def getKeywords(self, text):
        text = self.removePunctations(text)
        words = self.splitWords(text)
        words = self.removeStopWords(words)
        uniqueWords = list(set(words))

        keywords = [{'word': word, 'count': words.count(word)} for word in uniqueWords]
        keywords = sorted(keywords, key=lambda x: -x['count'])

        return (keywords, len(words))

    def getSentenceLengthScore(self, sentence):
        return (self.ideal - abs(self.ideal - len(sentence))) / self.ideal

    # Jagadeesh, J., Pingali, P., & Varma, V. (2005). Sentence Extraction Based Single
Document Summarization. International Institute of Information Technology, Hyderabad,
India, 5.
    def getSentencePositionScore(self, i, sentenceCount):
        normalized = i / (sentenceCount * 1.0)

        if normalized > 0 and normalized <= 0.1:
            return 0.17
        elif normalized > 0.1 and normalized <= 0.2:
            return 0.23
        elif normalized > 0.2 and normalized <= 0.3:
            return 0.14
        elif normalized > 0.3 and normalized <= 0.4:
            return 0.08
        elif normalized > 0.4 and normalized <= 0.5:
```

```
            return 0.05
        elif normalized > 0.5 and normalized <= 0.6:
            return 0.04
        elif normalized > 0.6 and normalized <= 0.7:
            return 0.06
        elif normalized > 0.7 and normalized <= 0.8:
            return 0.04
        elif normalized > 0.8 and normalized <= 0.9:
            return 0.04
        elif normalized > 0.9 and normalized <= 1.0:
            return 0.15
        else:
            return 0

    def getTitleScore(self, title, sentence):
        titleWords = self.removeStopWords(title)
        sentenceWords = self.removeStopWords(sentence)
        matchedWords = [word for word in sentenceWords if word in titleWords]

        return len(matchedWords) / (len(title) * 1.0)

    def splitSentences(self, text):
        tokenizer = nltk.data.load('file:' +
os.path.dirname(os.path.abspath(__file__)).decode('utf-8') + '/trainer/english.pickle')

        return tokenizer.tokenize(text)

    def splitWords(self, sentence):
        return sentence.lower().split()

    def removePunctations(self, text):
        return ''.join(t for t in text if t.isalnum() or t == ' ')
```

```
def removeStopWords(self, words):
    return [word for word in words if word not in self.stopWords]

def getStopWords(self):
    with open(os.path.dirname(os.path.abspath(__file__)) + '/trainer/stopWords.txt') as file:
        words = file.readlines()

    return [word.replace('\n', '') for word in words]
```

After these keywords are obtained, an accurate summary of the entire conversation is compiled, and is available to the user on an on-demand basis.

TESTING

CHAPTER 5

TESTING

Testing is a major part of any software product life cycle. The purpose of testing is to establish the presence of defects in the software and is usually the most time-consuming part of the software development process. According to most estimates testing requires 30 to 50 percent of the total development effort. There are many testing procedures for software depending on the application. This section describes the testing procedure adopted in the project and the result of the tests.

5.1 TESTING PROCESS

The most widely used testing processes consist of different stages. The testing process of this assignment is based on four stages. They are:

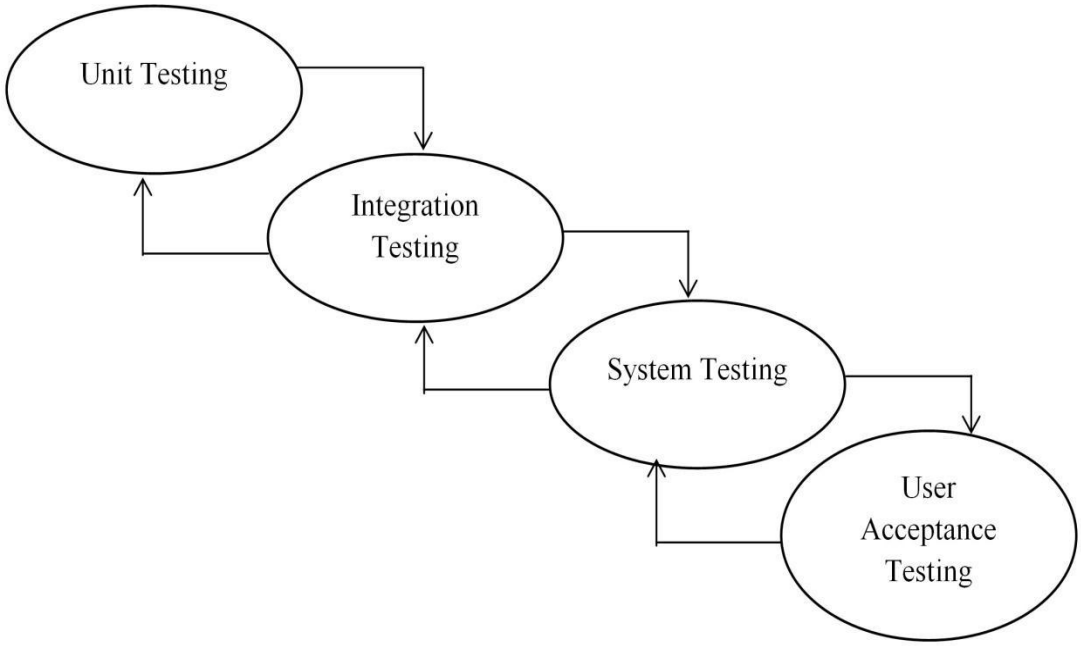

Figure 5.1: Testing Process

5.1.1 UNIT TESTING

Unit testing is essential for the verification of the code produced during the coding phase and hence the goal is to test the internal logic of the

components. Individual components are tested to ensure that they operate correctly. Each component is tested independently without other system components.

The individual components may be the Reminder module and the Summarizer module. The display of each component was tested separately here to verify that desired results were achieved. The testing was carried out during the coding phase.

Table 5.1: Unit Testing

Test Case:	1
Name of Test:	Testing of IMA application
Items Being Tested:	Messaging module reminder module, summarizer module
Expected Output:	Message sent successfully, reminder set successfully, summary displayed successfully
Actual Output:	Message sent successfully, reminder set successfully, summary displayed successfully
Remarks:	Test Pass

5.1.2 INTEGRATION TESTING

The process of system integration involves building a system from its components and testing the resultant system for problems that arise from component interactions. The components that are integrated may be off-the-shelf components, reusable components that have been adapted for a particular system or newly developed components. For many large systems, all three types of components are likely to be used. Integration testing checks that these components actually work together, are called correctly and transfer the right data at the right time across their interfaces.

The integration testing in this project comprised of combining and testing the related components of the Messaging assistant together as a whole. The

Android application, Reminder module, and summarizer module were integrated and tested together as an entire component. This testing was carried out upon completion of the Android application.

Table 5.2: Integration Testing

Test Case:	2
Name of Test:	Compilation of Source Code
Item Being Tested:	Source Code
Expected Output:	Application working as expected with 0 errors
Actual Output:	Application working as expected with 0 errors
Remarks:	Test Pass

5.1.3 SYSTEM TESTING

The different components are integrated to make up the entire system. The testing process is concerned with finding errors, which result from unanticipated interactions between the different components. It is also concerned with validating that the system meets its functional and non-functional requirements.

This testing procedure was carried out iteratively with addition of every new component. The initial component was the creation of the android application. This was system-tested with the addition of reminder module and the summarizer module.

Table 5.3: System Testing

Test Case:	3
Name of Test:	Checking the application for error
Item Being Tested:	Source Code
Expected Output:	IMA working successfully with 0 errors
Actual Output:	IMA working successfully with 0 errors
Remarks:	Test Pass

5.1.4 USER ACCEPTANCE TESTING

User acceptance of a system is the key factor for the success of any system. The system under consideration is tested for user acceptance by constantly keeping in touch with the prospective system users at the time of developing and making changes wherever required. The system developed provides a friendly user interface that can easily be understood even by a person who is new to the system.

Table 5.4: User Acceptance Testing

Test Case:	4
Name of Test:	Checking the application for error
Item Being Tested:	Source Code
Expected Output:	IMA working successfully with 0 errors
Actual Output:	IMA working successfully with 0 errors
Remarks:	Test Pass

5.2 TESTED ENVIRONMENT

Test Environment consists of elements that support test execution with software, hardware and network configured. Test environment configuration must mimic the production environment in order to uncover any environment/configuration related issues.

Table 5.5: Tested Environment

Sl. No.	Test Cases	Result	Remarks
1	User Sign-up	Allows user to register to our application	Successful
2	User Login	Allows user to login if the user is registered	Successful
3	Contact List Display	Displays the contact list of that particular user	Successful, when the user logs in giving the correct credentials

4	Messaging	Message received at the other end.	Successful
5	Reminder Setting	If any imperative or any ordering statements occurs.	Successful
6	Alarm	Alarm is set for that particular imperative sentence.	Successful
7	Summary Viewer	Accurate summary is shown if the user requests for it.	Successful

SNAPSHOTS

CHAPTER 6

SNAPSHOTS

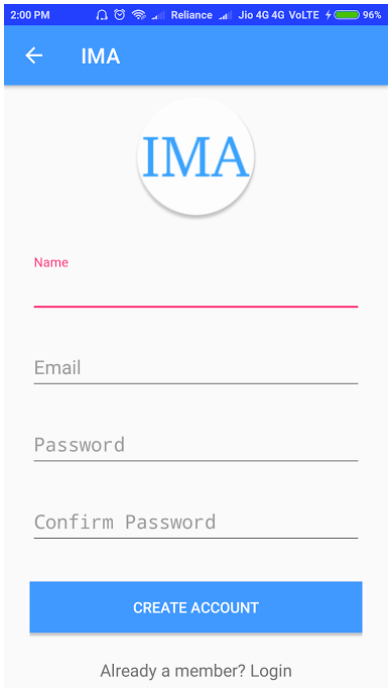

Figure 6.1: Signup Page

This shows the Signup page of the application through which the user should enter his valid credentials and register for our application.

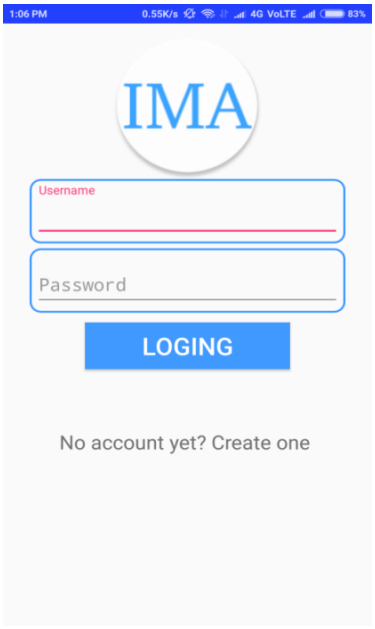

Figure 6.2: Login Screen

This shows the login screen of our application through which the user has to input his/her valid credentials for proceeding to the next screen.

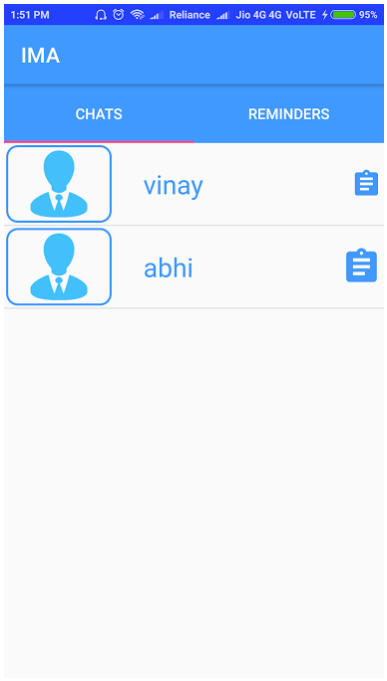

Figure 6.3: Contact List

This shows the contact list of the particular user after his/her login. If login fails, they will be prompted to enter their credentials again correctly.

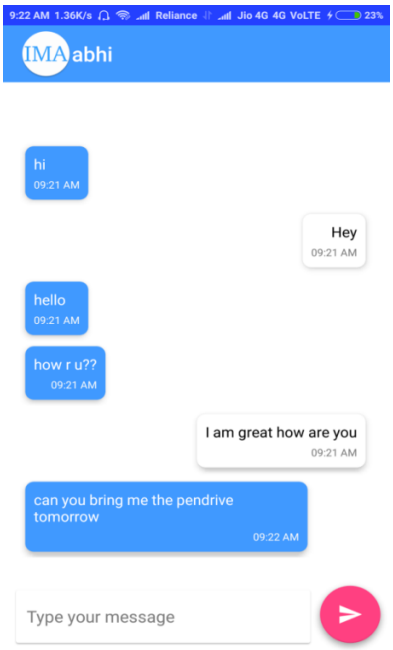

Figure 6.4: Conversation Between Two Users

This is a sample of the conversation between two users. The white colored box of message indicates one user and the blue colored box indicates the other user who is messaging in the receiving end.

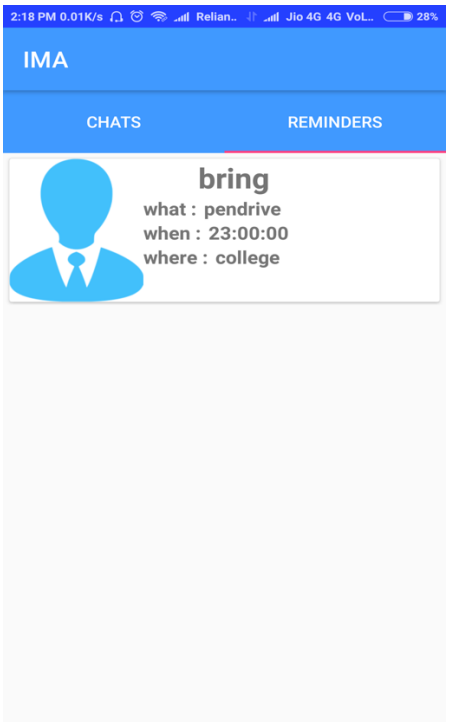

Figure 6.5: Reminder Being Set

This shows how the reminder set by our android application by analysing the conversation between the two users. This includes what he has to do, when he has to do, where he has to do.

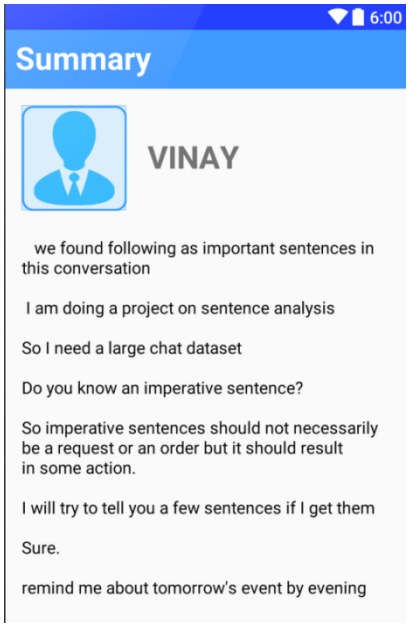

Figure 6.6: Summary of the Conversation

This shows the summary of the particular conversation which user has requested. This analyses the entire conversation between user and the requested user, and after analysing it, shows the required output i.e. the summary between them.

OBSERVATIONS AND RECOMMENDATIONS

CHAPTER 7

OBSERVATIONS AND RECOMMENDATIONS

The proposed system will absolve the user from keeping track of important aspects in any conversation, many of which are deemed important and forgotten otherwise. This system is most helpful, and is recommended for people who tend to forget things on a daily basis, absent-minded people, and is helpful to people in general.

The following can be considered as advantages of the proposed system:

- The application serves as a messenger, and will facilitate with the communication with other users, i.e., separate applications for messaging and reminder setting is not required.
- The application is android based, which occupies majority of the mobile market space. Hence, many customers can have easy access to the application.
- The users are notified of the reminders in the form of alarms as well, and thus the user does not have to go through all the reminders set in the “Reminders” menu.
- The reminders are detailed. It tells the user what, where, when, and who about the context in question.
- The user interface is made as simple as possible to minimize the learning curve for the user, i.e., the user can operate the application with little-to-no effort, right out of the box.
- All aspects of the application are handled using online resources such as APIs and servers, thereby reducing the resource overhead caused by the application on the user’s smartphone.

Certain drawbacks of the application could be:

- Since most of the application resources are online, a constant internet connection, via Wi-Fi or mobile data is necessary for the application to function at its best.
- Chatting with another user also requires a continuous internet connection.
- Given the application is android based, both the users are expected to have an android device to install and utilize the application.

FUTURE ENHANCEMENTS

CHAPTER 8

FUTURE ENHANCEMENTS

- The current application aims to summarize and set notifications in a two-person conversation, and so, context recognition and summarization of group chats can be added as a feature.
- Summarization and reminder setting can be implemented for voice messaging, voice calling, and video calling.
- The application currently works with English as its main language, and other regional languages (such as Hindi or Kannada) or foreign languages (such as French, German, etc.) can be implemented as well.
- The application can also be improved to act as an assistant for users, similar to Google Allo, which answers user's questions and performs some task on behalf of them.
- Since application detects eventful sentences and task oriented messages, it can be enhanced to schedule day-to-day tasks for users.

CONCLUSIONS

CONCLUSIONS

This project proposes a new concept, design and implementation of a summarizer and reminder notification. The application is suited for, and recommended for all users who may or may not have a tendency to forget things on a regular basis. The application aims to serve as a small extension that will assist the user in the long term. The advantages and drawbacks of the proposed system have been discussed in the previous section. Advancements in computers and information technology has made intelligent systems a reality, such as extracting information from natural human language. In this paper, an idea is presented for enhancing the user experience in communication using instant messaging on mobile devices. The proposed model is feasible and hopefully will be working towards elimination of the existing inconvenience. In the future, with various newly introduced concepts and protocols in the growing field of Mobile Computing and Natural Language processing, it could be possible to overcome the impediments, and hence produce an impeccable model.

REFERENCES

[1] Log-Based Chat Room Monitoring Using Text Categorization: A Comparative Study Eiman M. Elnahrawy, Department of Computer Science, University of Maryland, College Park, MD 20742, USA http://paul.rutgers.edu/~eiman/eiman02monitoring.pdf

[2] A SURVEY OF TEXT SUMMARIZATION TECHNIQUES, Ani Nenkova, University of Pennsylvania, Kathleen McKeown, Columbia University https://www.cs.bgu.ac.il/~elhadad/nlp16/nenkovamck

[3] A Survey on Automatic Text Summarization, Dipanjan Das Andr´e F.T. Martins Language Technologies Institute, Carnegie Mellon University, November 21, 2007 https://www.cs.cmu.edu/~afm/Home_files/Das_Martins_survey_summarization.pdf

[4] Summarizing Spoken and Written Conversations Gabriel Murray and Giuseppe Carenini, Department of Computer Science, University of British Columbia, Vancouver, BC V6T 1Z4, Canada http://www.aclweb.org/anthology/D08-1081.

[5] Multi-Dimensional Fragment Classification in Biomedical Text ByFengxia Pan, A thesis submitted to the School of Computing in conformity with the requirements for the degree of Master of Science Queen's University Kingston, Ontario, Canada September 2006 FengxiaPanThesis.pdf.

[6] Yoon Kim New York University: http://www.aclweb.org/anthology/D14- 1181.

[7] Experiments with Sentence Classification Anthony Khoo, Yuval Marom and David Albrecht Faculty of Information Technology, Monash University Clayton, VICTORIA 3800, AUSTRALIA http://www.aclweb.org/anthology/U06-1005.

[8] Call Centre Conversation Summarization: A Pilot Task at Multiling 2015, Benoit Favre, EvgenyStepanov, Jeremy Trione, Frederic Bechet, Giuseppe Riccardi, Aix-Marseille University, CNRS, LIF UMR 7279, Marseille, France, University of Trento, Via Sommarive 5, Trento, Italy.

[9] XMPP protocol documentation http://www.xmpp.org

[10] Analysis and Detection of Eventful Messages in Instant Messenger, Abhijit R. Joshi, Darshana Desai, Karan Shah, Chintan Shah.

[11] TextTeaser Summariation, Jolo Balbin, 2014.

NaCoNIM - 2017, May 10th - 12th ISBN:978-81-929425-1-3

INTELLIGENT MESSAGING ASSISTANT

Nikhil N Prasad[1], Vilvek V[1], Nikhil Gupta N C[1], Vinayaka K[1], Dept of CSE, KSIT
Harshavardhan J R[2], Associate Professor, Dept of CSE, KSIT

Abstract—**One of the most common problems we face every day is that, people often forget minor, but crucial jobs we were requested to perform, such as calling someone at a later time, mailing an important document etc. People who forget to do so often feel dissatisfied that they did not perform the assigned task. The aim of this project is to create a messaging application that recognizes the context of its users conversation, stores it, analyses the conversations, determines the Action -Driving Statements, and triggers a reminder to its user, thereby informing him/her to carry out the task decided during the conversation. Integration and possibilities of the IMA in real world scenarios are considered and described in this paper. The possible design and implementation of an autonomous reminder and conversation summarizer is presented and described. The development frameworks as well as technologies that are used to realize this concept is described. The concept of text summarization and detection of plausible reminders implementable using machine learning is also described.**

Keywords: Automatic conversation summarizer, Instant Messaging (IM), XMPP, Natural Language Processing, Machine Learning.

I. Introduction

The advancement of the Internet and the smartphone have changed the way people communicate in this era. One of the biggest advancements due to the advent of these technologies is that most conversations today happen through electronic mail and instant messaging (IM). Instant messaging (IM) is a set of communication technologies used for text-based real time communication between two or more participants over the Internet. Often referred to as "chat", IM offers real-time communication between two or more users. IM solutions, today, include group chat, conferencing, voice, video, and much more. IM is highly popular with both consumer and enterprise users. This new trend of conversing through Instant messaging(IM) has opened up an opportunity to analyse the conversation data and build useful applications using them. One such application of conversation analysis is devising a solution to the aforementioned problem.In recent years, the number of smartphone users have surpassed the number of PC users, thanks to the portability and convenience of prompt information access. This has encouraged advancements in mobile computing and the popularity of mobile operating systems such as Android. Since this advent of smartphones, the general messaging paradigm has shifted drastically to these handheld devices. worldwide IM user accounts are expected to grow from over 5.8 billion in 2017 to over 8.3 billion by year-end 2021, representing an average annual growth rate of 9% Every smartphone users uses IM and it is found that Mobile users spent more than 10% of their mobile usage time on Social Messaging Apps Stephen Wang, a senior PM at WeChat said "At every time throughout the day, there is a touchpoint between WeChat and your normal life." Thus a large portion of casual communication happens through IM on a daily basis, which often contains sentences that assigns a job responsibility to either of the communication participants. For example, in a conversation between students, student ***x*** may text student ***y*** to get document **z** to college. We often have a tendency to forget these tiny responsibilities, and thus, an automatic reminder about this responsibility would be helpful to user. IM also contains group chats. Typically, these chat rooms are always filled with conversations and not all chats are important to all the users. Thus, a summary of these conversations will ensure that the user will not miss out on any important messages in the group. For example, an IM group containing student participants, which would normally comprise much of general conversations, would also have traces of messages regarding placements during placement season. In such a scenario, a student may fail to notice a vital placement intimation due to an abundance of mundane messages. This could prove to create a gulf in communication. One of the ways to bridge this gap is to automatically create a summary of the conversation, which provides the user with the important messages. This projects aims in providing reminders for eventful messages and a conversation summary of instant messages. The software consists of a chat paradigm, meaning a user can communicate with another user via text message (similar to popular messaging services such as WhatsApp, Facebook Messenger, and so on). The improvement that our application boasts, is the reminder and summarizer feature. It keeps a detailed summary of the conversation between the users in real-time, and automatically detects the context of the conversation. The application is always on the look-out for anything that resembles an imperative sentence. Once it does, a reminder is added to the

database maintained by the application, and a notification is set at the given time to the user. The user can also go through a history of all his/her reminders within the application itself. This describes the basic functionality of the proposed application. The rest of the paper is described as follows. Section II describes the design , section III describes system architecture, section IV describes observations and recommendations, section V describes future enhancements, section V1 provides the conclusion and section VII gives the Acknowledgement section VIII describes the references used.

II. Design

Most of the existing Instant Messaging applications do not give personal conversation via an API call, as it is matter of privacy, and even if they do provide, it is not flexible for analysis of plausible reminders and summarization of conversation in real time. Thus, we have to develop our own Instant Messaging application prototype which we have called Intelligent Messaging Assistant (IMA). The IMA should provides three main functionality for users:

a) Instant Messaging

b) Reminders setting

c) Conversation summarization

a) Instant Messaging

The app will provide one on one and group conversation. We restrict the conversation to be textual-based only for this project. We use a widely adopted open protocol for instant messaging, Extensible Messaging and Presence Protocol (XMPP). XMPP is a communications protocol for message-oriented middleware based on XML (Extensible Markup Language)[9]. It enables near-real-time exchange of structured yet extensible data between two or more network entities. Since XMPP is an open standard, many implementations of the server, client, and the library are available and so, we use openfire as XMPP server for instant messaging. Openfire is a real time collaboration (RTC) server licensed under the Open Source Apache License. The IMA is developed using the spark library.

b) Reminder setting

In the scope of this project, eventful sentences are those which give or request some task or responsibility to one of the participants of the conversation. We use machine learning binary classification algorithm for finding whether the given sentence qualifies as a candidate for an eventful sentence. The input to this algorithm is a message and the output is whether the given sentence is a reminder sentence (positive classification) or not (negative classification). The algorithm for determining reminders is described as follows:

- A conversation often contain introductory phrases such as “hey”, ”hello”, ”whats up?”, ”how are you”, and so on. These messages are not considered for plausible reminder analysis.
- Stop words such as “ah”,”oh”,”sup” etc are removed from the message.
- It is known from experience that the eventful sentences will have at least 10 characters in it. Thus, any message containing lesser than 10 characters are not considered for plausible remainder analysis.
- The sentences that pass all of the the above condition steps are considered for plausible remainder analysis and is fed to classifier algorithm
- If the output of the classifier is positive i.e the sentence is a eventful sentence, it is set as a reminder.
- The degree of correctness in classification achieved by the algorithm depends on feature selection. Feature selection therefore plays an important role part in this project. The following are different features useful in sentence classification:
- Predefined Phrases and Keywords
- Sentences containing words, phrases, or keyword such as ‘remind me’, ‘remember’, ‘ping’, ‘notify’, ’get’ etc. are strong candidates for being classified as a reminder. One of the features checks for the existence of such words.
- Adverb-Verb Dependency or Imperative Sentence
- Since many imperative sentences will have an adverb followed by a verb, this feature is the most useful for our classifier algorithm. For example, in the sentence please bring pen drive tomorrow. Here please is the adverb and bring is the verb.
- Existence of location and time with a verb in sentence. Messages that have time, date, location, or a combination of any of these three characteristics are likely to be candidates for a reminder.

4) Conversation Summarizer: Summarization has become an important and timely tool for assisting and interpreting information in today’s fast-growing information age. Summarization focuses on building extractive, generic, and surface-level-feature-based summarizers. These extractive summarizers select and present pieces of the original speech transcripts or audio segments as summaries, rather than rephrase or rewrite them. The output summary could be textual (transcripts) or spoken (e.g., concatenated audio clips). In our project we summarize the conversation transcripts which often contains filler words. Thus removing these filler words in the transcripts is the first step in summarizer. The pieces to be extracted from the transcripts can be done by assigning weights to each sentences based on certain features and methodologies. The features that can be considered for extractive summarization are as follows.

Content Word (Keyword) Feature:

Content words or Keywords are usually nouns and determined using tf × idf measure. Sentences having keywords are of greater chances to be included in the summary.

Sentence Length Feature

Very short sentences are not included in the summary.

Cue-Phrase Feature

Sentences containing any cue phrases (e.g. “in conclusion”, "in short", “this report”, “summary”, “argue”, “purpose”, “develop”, “attempt” etc.) are most likely to be in the summary.

The summarization algorithm uses basic summarization features and build from it. Those features are: title feature, sentences length, sentence position, and keyword frequency.

- Title feature is used to score the sentence with the regards to the title.
- Sentence length is scored depends on how many words are in the sentence.
- Sentence position is where the sentence is located. The introduction and conclusion will have higher score for this feature.
- Keyword frequency is just the frequency of the words used in the whole text.

The algorithm works as follows:

- The entire conversation is fed to the algorithm in the form of a file (a single file is more convenient to handle longer conversations or large bodies of text). The features are determined by the summarizer function by implementing the **parser module.** This module uses the natural language toolkit (NLTK), which works towards analysing and interpreting the languages humans use naturally.
- The NLTK functions enables the parser to separate every sentence in the summary file, split individual words from each sentence, remove punctuations (as they are not necessary), determine and remove stop words (present at the end of typical sentences), determine keywords, and send these to the **summarizer function**.
- The **summarize function** handles the summarization of the algorithm. The title and the input file is fed into the function as input parameters. The summarizer function determines the title feature by finding out the sentence score, sentence length, sentence position and keyword frequency. The function appends these parameters which is sent as the final summary.
- The final summary is then compiled and is ready to be presented to the user on his/her request.

III. System Architecture

The proposed system architecture contains three main modules: Instant Messaging (android client and openfire server), Reminder, and Summarizer modules. The Instant Messaging client is further divided into three sub modules: Reminder Alarm List, Messaging, and Summary List (Figure1).

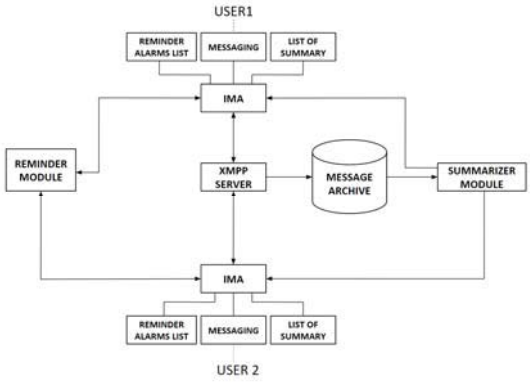

Figure 1-System Architecture

The working of our model can be broken down into a series of steps as follows (implementation of the steps may not be in the same order always) :

1. The user logs in to the IMA application (Instant Messaging client) using his username and password(figure 2). If he/she does not have one, he/she can create an account.(figure 3)
2. The user is then met with the chat selection screen; here, he/she can select the recipient with whom he would like to communicate. (figure 4)
3. The XMPP server(Instant Messaging Server) transfers message between users.(figure 5) and also archives the message in Message Archive database
4. whenever IMA receives the message the Reminder Alarm List sub module of IMA send it to Reminder module
5. The Reminder module is actually a api calls to ***Luis (by Microsoft)*** or to ***API.AI***(google) which are language understanding systems. These APIs are initially trained using a training data set to be on the look-out for words or phrases that might be a viable candidate for a reminder.
6. The APIs implement **machine learning algorithms**, and hence, are capable of learning ways in which a word or a phrase can be classified as a reminder candidate, (say, by looking-out for synonyms of the previously defined words from the training data set). By

doing so, the API becomes increasingly accustomed to the user's style of conversation and will be better able to determine potential reminders in the future.

7. When a sentence has been classified as a candidate for a reminder, it is broken down in order to determine the various parameters of the reminder, such as the subject, the object, the action, the location, date-time parameters, and so on.

8. Once the parameters have been determined, they are sent back to the IMA application ,where the Reminder Alarm List module set these reminders as alarms and is also made available for the user as a list.(Figure 6)

9. Meanwhile, the user can ask for summary of his conversation by clicking a button on respective Contact List in messaging module, which triggers the Summary List sub module of IMA. The Summary List sub module makes request for remote Summarizer module for summary.

10. The Summarizer module retrieves the conversation from message archive and apply summarizer algorithm as described in the previous section. The result is sent back to the IMA application where its is displayed to the user.(Figure 7)

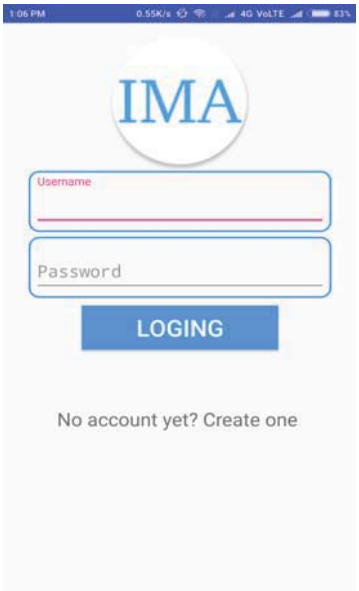

Figure 2- Login Screen of user in application

Figure 3 - Signup Page of the Application

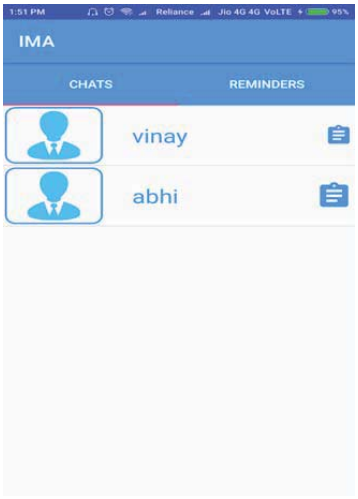

Figure 4- Contact list

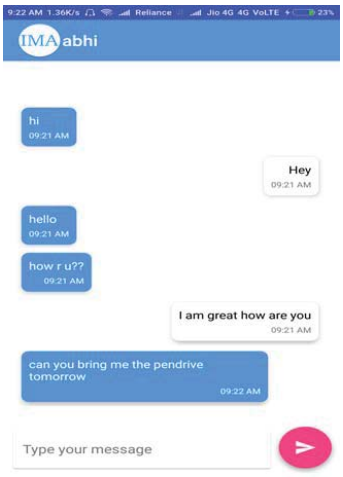

Figure 5- Example of a conversation

 ISBN:978-81-929425-1-3

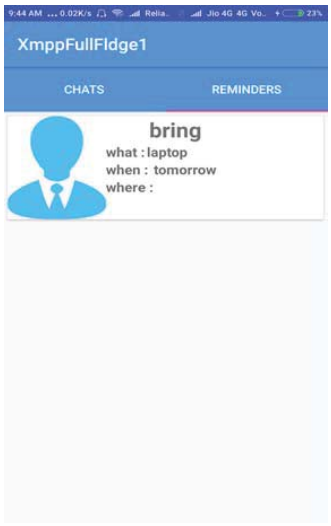

Figure 6 - Reminder being Set in the application

Figure 7 - Summary of the conversation

IV. Observations and Recommendations

The proposed system will absolve the user from keeping track of important aspects in any conversation, many of which are deemed important and forgotten otherwise. This system is most helpful, and is recommended for people who tend to forget things on a daily basis, absent-minded people, and is helpful to people in general.

The following can be considered as advantages of the proposed system:

1. The application serves as a messenger, and will facilitate with the communication with other users, i.e., separate applications for messaging and reminder setting is not required.
2. The application is android based, which occupies majority of the mobile market space. Hence, many customers can have easy access to the application The users are notified of the reminders in the form of alarms as well, and thus the user does not have to go through all the reminders set in the “Reminders” menu.
3. The reminders are detailed. It tells the user what, where, when, and who about the context in question.
4. The user interface is made as simple as possible to minimize the learning curve for the user, i.e., the user can operate the application with little-to-no effort, right out of the box.
5. All aspects of the application are handled using online resources such as APIs and servers, thereby reducing the resource overhead caused by the application on the user’s smartphone.
6.

Certain drawbacks of the application could be:

1. Since most of the application resources are online, a constant internet connection, via Wi-Fi or mobile data is necessary for the application to function at its best.
2. Chatting with another user also requires a continuous internet connection.
3. Since the application is android based, both the users are expected to have an android device to install and utilize the application.

V. Future Enhancements

- The current application aims to summarize and set notifications in a two-person conversation, and so, context recognition and summarization of group chats can be added as a feature.
- Summarization and reminder setting can be implemented for voice messaging, voice calling, and video calling.
- The application currently works with English as its main language, and other regional languages (such as Hindi or Kannada) or foreign languages (such as French, German, etc.) can be implemented as well.
- The application can also be improved to act as an assistant for users, similar to Google Allo, which answers user's questions and performs some task on behalf of them.
- Since application detects eventful sentences and task oriented messages, it can be enhanced to schedule day-to-day tasks for users.

VI. Conclusion

We propose a concept, design and implementation of a summarizer and reminder notification. The application is suited for, and recommended for all users who may or may not have a tendency to forget things on a regular basis. The application aims to serve as a small extension that will assist the user in the long term. The advantages and drawbacks of the proposed system have been discussed in the previous section. Advancements in computers and information technology has made intelligent systems a reality, such as extracting information from natural human language. In this paper, we presented an idea for enhancing the user experience in communication using instant messaging on mobile devices. The proposed model is feasible and we are hopeful in working towards eliminating this inconvenience. In the future, with various newly introduced concepts and protocols in the growing field of Mobile Computing and Natural Language pro-

cessing, it could be possible to overcome the impediments, and hence produce a impeccable model.

VII. Acknowledgement

The Authors would like to thank VGST (Vision Group on Science and Technology), Government of Karnataka, India for providing infrastructure facilities through the K-FIST Level I project at KSIT, CSE R&D Department,Bengaluru

VIII. References

[1] Log-Based Chat Room Monitoring Using Text Categorization: A Comparative Study Eiman M. Elnahrawy, Department of Computer Science, University of Maryland, College Park, MD 20742, USA http://paul.rutgers.edu/~eiman/eiman02monitoring.pdf

[2] A SURVEY OF TEXT SUMMARIZATION TECHNIQUES, Ani Nenkova, University of Pennsylvania, Kathleen McKeown, Columbia University https://www.cs.bgu.ac.il/~elhadad/nlp16/nenkovamck

[3] A Survey on Automatic Text Summarization, Dipanjan Das Andr´e F.T. Martins Language Technologies Institute, Carnegie Mellon University, November 21, 2007 https://www.cs.cmu.edu/~afm/Home_files/Das_Martins_survey_summarization.pdf

[4] Summarizing Spoken and Written Conversations Gabriel Murray and Giuseppe Carenini, Department of Computer Science, University of British Columbia, Vancouver, BC V6T 1Z4, Canada http://www.aclweb.org/anthology/D08-1081.

[5] Multi-Dimensional Fragment Classification in Biomedical Text By Fengxia Pan, A thesis submitted to the School of Computing in conformity with the requirements for the degree of Master of Science Queen's University Kingston, Ontario, Canada September FengxiaPanThesis.pdf.

[6] Yoon Kim New York University: http://www.aclweb.org/anthology/D14- 1181.

[7] Experiments with Sentence Classification Anthony Khoo, Yuval Marom and David Albrecht Faculty of Information Technology, Monash University Clayton, VICTORIA 3800, AUSTRALIA http://www.aclweb.org/anthology/U06-1005.

[8] Call Centre Conversation Summarization: A Pilot Task at Multiling 2015, Benoit Favre, Evgeny Stepanov, Jeremy Trione, Frederic Bechet, Giuseppe Riccardi, Aix-Marseille University, CNRS, LIF UMR 7279, Marseille, France, University of Trento, Via Sommarive 5, Trento, Italy.

[9] XMPP protocol documentation http://www.xmpp.org

[10] Analysis and Detection of Eventful Messages in Instant Messenger, Abhijit R. Joshi, Darshana Desai, Karan Shah, Chintan Shah.

Kammavari Sangham (R) - 1952
K.S.INSTITUTE OF TECHNOLOGY
(Affiliated to VTU, Belagavi & Approved by AICTE, New Delhi)
No.14, Raghuvanahalli, Kanakapura Main Road, Bengaluru - 560109
Tel : +91-80-28435722 / 24 Fax : +91-80-28435723 Email : principal.ksit@gmail.com Website : www.ksit.ac.in

Kammavari Sangham (R) 1952 — 60 Years — 1952-2012

Department of Computer Science and Engineering

National Conference on Networking, Image Processing & Multimedia

NαCoNIM

May 10th - 12th, 2017

CERTIFICATE

This is to certify that NIKHIL GUPTA M.C *of*

K. S. Institute of Technology presented a paper titled INTELLIGENT MESSAGING ASSISTANT

Harshavardhan J.R.
Chief Co-ordinator

Dr. Rekha B. Venkatapur
HOD / Convener

Dr. T.V. Govindaraju
Principal / Director

Kammavari Sangham (R) - 1952

K.S.INSTITUTE OF TECHNOLOGY

(Affiliated to VTU, Belagavi & Approved by AICTE, New Delhi)

No.14, Raghuvanahalli, Kanakapura Main Road, Bengaluru - 560109

Tel : +91-80-28435722 / 24 Fax : +91-80-28435723 Email : principal.ksit@gmail.com Website : www.ksit.ac.in

Kammavari Sangham (R) 1952 — 60 Years — 1952-2012

Department of Computer Science and Engineering

National Conference on Networking, Image Processing & Multimedia

NαCoNIM

May 10th - 12th, 2017

CERTIFICATE

This is to certify that NIKHIL.N.PRASAD *of*

K. S. Institute of Technology *presented a paper titled* INTELLIGENT MESSAGING ASSISTANT

Harshavardhan J.R.
Chief Co-ordinator

Dr. Rekha B. Venkatapur
HOD / Convener

Dr. T.V. Govindaraju
Principal / Director

Kammavari Sangham (R) - 1952

K.S.INSTITUTE OF TECHNOLOGY

(Affiliated to VTU, Belagavi & Approved by AICTE, New Delhi)

No.14, Raghuvanahalli, Kanakapura Main Road, Bengaluru - 560109

Tel : +91-80-28435722 / 24 Fax : +91-80-28435723 Email : principal.ksit@gmail.com Website : www.ksit.ac.in

Kammavari Sangham (R) 1952 — 60 Years — 1952-2012

Department of Computer Science and Engineering

National Conference on Networking, Image Processing & Multimedia

NαCoNIM

May 10th - 12th, 2017

CERTIFICATE

This is to certify that VINAYAKA. K *of*

K. S. Institute of Technology *presented a paper titled* INTELLIGENT MESSAGING ASSISTANT

Harshavardhan J.R.
Chief Co-ordinator

Dr. Rekha B. Venkatapur
HOD / Convener

Dr. T.V. Govindaraju
Principal / Director

Kammavari Sangham (R) - 1952

K.S.INSTITUTE OF TECHNOLOGY

(Affiliated to VTU, Belagavi & Approved by AICTE, New Delhi)

No.14, Raghuvanahalli, Kanakapura Main Road, Bengaluru - 560109

Tel : +91-80-28435722 / 24 Fax : +91-80-28435723 Email : principal.ksit@gmail.com Website : www.ksit.ac.in

Kammavari Sangham (R) 1952 — 60 Years — 1952-2012

Department of Computer Science and Engineering

National Conference on Networking, Image Processing & Multimedia

NαCoNIM

May 10th - 12th, 2017

CERTIFICATE

This is to certify that VIVEK.B *of*

K. S. Institute of Technology *presented a paper titled* INTELLIGENT MESSAGING ASSITANT

Harshavardhan J.R.	**Dr. Rekha B. Venkatapur**	**Dr. T.V. Govindaraju**
Chief Co-ordinator	*HOD / Convener*	*Principal / Director*