Abstract

This project presents the design and implementation of an intelligent banking assistant with capabilities of executing customer's queries given in text and providing output in textual as well as in voice format. We implement two approaches in building the banking assistant - one using Natural Language Processing (NLP) techniques and other using AIML, a popular language to implement assistants (sometimes also called, chatterbots).

Natural Language Processing involves various techniques to analyze and process text for variety of applications like information retrieval, question-answering, speech recognition, etc. In this approach, we will use some of those techniques to build a banking assistant which will try to provide responses to users by doing a in-depth analysis of user queries.

AIML is a XML-based language which is used for writing chatbots. It's quite popular and has been used in many chatterbots like Jabberwacky, ALICE, etc. It uses pattern-matching to find pre-defined patterns which match to the user's query and providing some response to the user based on some answer templates.

The aim of this project is to provide a user-friendly interface which will enable users to ask queries related to banking.

Acknowledgements

I would like to acknowledge everyone who supported me during the project, my team members who coordinated with me, all my friends who gave guidance related to the project and also supported me, our mentor Prof. Rajni Pamnani for continuous support during all the phases, our Head of the Department, Prof. Bharathi H. N. for providing all the necessary facilities and for supporting us.

- ***Kiner Shah***

I would like to thank and acknowledge each and every person related to our project and contributed for it in some way or other. Special thanks to Prof. Rajni Pamnani for guiding us through the entire project.

- Darshan Shah

The completion of this undertaking could not have been possible without the assistance and participation of my team members. I would like to express deep appreciation and indebtedness to Prof. Rajni Pamnani, our project mentor for endless support and understanding spirit during our project implementation. I gratefully acknowledge the contributions of my parents, friends and others who provided their support.

- ***Mohit Shetty***

Table of Contents

Chapter 1: Introduction

In this chapter, we first discuss about the motivation which led to the inception of this project. Then, we discuss a proper formal problem definition which our project will address. Then, we discuss the scope covered by our project.

1.1 Motivation

Banking is done by hundreds of people daily - some of them making transactions, some of them applying for loans, some of them depositing cheques, some of them checking their passbooks, some of them inquiring about the policies, some of them opening accounts, etc. But, in this process there are certain situations when customers face problems. For example, a man going to the bank for opening an account and he finds out that there are different types of accounts as alternatives (savings account, current account, fixed deposit account, etc.), but now he is confused as he came to the bank with one goal i.e. to open an account. To clear the confusion, he asks the bank clerk for that. Now, there are many people who want to communicate with the clerk and thus, there is a queue. The person has to wait in the queue just for asking a simple doubt! Now, it's his turn and he asks the clerk, but he didn't understood what clerk explained for the first time, so he asked again.

There are various problems with this. Firstly, the person has no idea about how to open an account and despite of information available on various websites. Maybe, the person doesn't know how to use the Internet. Secondly, the person had confusion and had to wait in a line just for asking the clerk about the problem. Thirdly, he didn't understand the explanation given by the clerk for the first time and had to ask for better explanation. Imagine the problem when a person has to go and issue a passbook and get it updated regularly. It will be a waste of time since he has to go every time and wait in the queue for getting his work done.

This gave us motivation to design an interface, which not only helps the person to easily do his work but also allows him to interact with the interface and clarify whatever doubts he/she has. By keeping in mind different kinds of users (Beginners, Intermediates, and Experts) and designing an interface, we aim to provide a simple to use interface.

1.2 Problem Definition

In this section, we discuss the problem definition of our project. We have already seen the motivation which led to the inception of the project.

Usually bank's policies are well documented. But, understanding the document itself is a very tedious task. It usually contains complicated clauses, confusing disclaimers, and jargon terms which are difficult to understand by the common man. It is as difficult as pulling a truck full of black soil. This problem can be solved by providing advice to the customers in very simple and concise sentences. This problem becomes severe in case of customers who are unlettered. To solve this, it is advisable to give the customers output advice through an anthropomorphic interface like voice. Also, due to language barrier, some customers feel uncomfortable to communicate with the bank clerks. This can be solved by providing customers with language choice, however, this can complicate implementations. So, the problem here is *lack of understanding*.

Banking is time consuming. If we want to submit an application for a new cheque book, there is a long line, where we have to wait until our turn comes. Similar thing happens when we want to update our passbooks. Due to advancements in technology, this all can be made digital. The problem is, that the process, is *time consuming* and there is *loss of crucial time* for banking customers.

Our aim to solve the problems mentioned above by using intelligent systems which are based on two approaches - NLP and AIML.

1.3 Scope

The project covers a broad scope wherein different outcomes are expected to be delivered. The project aims to use various technologies which will help in completion of several modules that are the part of the project. The project is expected to cover the following aspects within its scope:

1. Building a GUI which supports all types of users and which covers all the functionalities of the project.
2. Inclusion of basic banking services like checking passbook, applying for a new cheque book, and if possible, fund transfers.

3. Inclusion of inquiry service:
    a. Using text as the mode of input: Users will simply be able to type their queries.
    b. Using speech as the mode of input: Users will be able to connect a microphone and give their queries by speaking.
    c. Using text as the mode of output: Users will get the output in simple textual format.
    d. Using speech as the mode of output: Users will be able to listen to the output with a constraint that speaker is connected and operates on full volume.
4. Executing the basic services mentioned above by means of natural language commands (either text or voice).
5. Security to bank customers by providing necessary authentication and privacy.

Chapter 2: Literature Review

2.1 Banking Assistants

There are many projects implemented earlier for banking assistant. First system is Royal Bank of Scotland's "Luvo". Luvo, is able to understand questions and then filter through huge amounts of information in a split second before responding with the answer. If Luvo is unable to find the answer, it passes the query on to a member of staff who can solve more complex problems. It will support staff to help them answer customer queries more quickly and easily. It has to be trained when dealing with new subject matter, but crucially, it learns from its mistakes and its answers become more accurate over time. One problem with Luvo, however, is that it can interact only with bank staff and not directly with customers. Also, there is still the need for human experts despite having AI deployed.

Kasisto's AI (KAI) banking is a conversational AI platform which makes engaging with customers as natural as chatting. The AI has deep financial knowledge and banking is easy from mundane tasks to complicated tasks – it's basically sending text messages. It is a very fine tool and does most of the tasks which is required for general banking like checking for previous transactions, making payments, telling information about accounts and the main point is that it is always learning. It chats with its users in a friendly manner. It is a good implementation of banking assistant.

Another such assistant was suggested which was implemented as a web service (based on black-box approach) able to process multiple client requests simultaneously and which generated responses by using a data repository (based on AIML).

A study of an intelligent voice based recognition chatbot which was implemented using Client-Server architecture was done. It followed the Simple Object Access Protocol (SOAP) model where the responses from server were classified into two categories: data retrieval and information output. All the messages were formatted in XML and encapsulated in SOAP message packet.

Another review conducted involved the ways of extracting task oriented information from conversational dialogs. It involved the use of Utterance Normalization, Mining of sub tasks

and AIML conversion. Utterance Normalization is used to cluster semantically similar utterances together. We then extract the possible subtasks and convert them into AIML.

2.2 Techniques / Applications of Natural Language Processing

We studied various works by researchers on natural language processing techniques and applications. An example-based dialogue system was suggested where the system uses dialogue examples rather than using rules or probabilistic models. So when a user input is given, the system selects the similar situation from the previous dialogue examples. The advantages include reduction of human labour and domain independence. The system also uses a long-term memory to store specific features from the user's dialogue. The system uses POS Tagging for sentence matching, named entity recognition and main action execution which may require domain expert knowledge.

Another work presents various NLP techniques used in speech recognition and text-to-speech. It presents various techniques for speech synthesis like Text Normalization (which includes sentence segmentation, tokenization, removal of non-standard words like Mr., Dr., etc.), POS Tagging, Grapheme to phoneme conversion (which includes mapping of the smallest written unit to a group of sounds) and word stress. For speech recognition, the work signifies the importance of grammars especially Context-free grammars and also mentions that out of two techniques of making grammar rules - hand-crafted rules and derivation from statistical analysis on labelled corpus of data - the latter approach is mostly used and leads to formation of N-grams models.

Another work present a single architecture using a single convolutional neural network, which can do many NLP tasks including POS Tagging, NER, Chunking (also called, partial or shallow parsing), semantic role labelling, etc and which is trained on these tasks using a weight sharing (an instance of multi-task learning). This leads to overcoming of several problems like freedom from hand-engineered rules, no propagation of errors from learning the cascaded features, removing the shallowness of the systems due to linearity of the classifiers.

Other than applications, there has been lots of research on NLP techniques and how to improve them. Some works show the improvements dependency parsing by considering the possibility of non-projective dependency parse trees being generated and try to solve them using spanning tree algorithms. An arc in the dependency tree, (w_i, r, w_j) is said to be

projective if and only if $w_i \rightarrow w_k$for all $i < k < j$ when $i < j$ and for all $j < k < i$ when $j < i$. Most of the dependency parsers deal with projective trees and not non-projective trees especially those for English as the language as English doesn't have many sentences that lead to non-projective arcs. Such graph based parsers are often more accurate than other parsers. Some other works have tried to improve the simple shift-reduce dependency parsers to give more accuracy by introducing new operations in the parser.

Other than parsing, some works have tried to improve the normalization process which gives input to other complex tasks like text-to-speech. This is done for text messages. The tasks include removal of unnecessary vowels, clipping of prefixes and suffixes, some special cases and finally conversion of words into abbreviations which are later used for efficient text-to-speech processing.

2.3 Artificial Intelligence Markup Language (AIML)

Improvements in Machine Learning Techniques and Data Mining has led to better decision making and robust processing tools standards like XML, thereby chatbots have become more practical in Day-to-Day applications for e.g. automatic telephone answering machine, information retrieval tools. One such study used is an ALICE system which was first used to help Chinese university students practice their conversational English skills. The study was focussed more on user attitudes rather than chatbot efficiency and used pre-existing conversational English skills. In all of these conversational entities one thing was common, that is, they were having the difficulty of maintaining dialogue for sustainable period of time.

Another study focussed on using ALICE as course enhancement tool with Social & political theory knowledge. ELIZA was the first famous chatbot. Dialog systems can sufficiently carry out the conversations with the users and can log the conversation which can be a good source for acquisition of Domain specific knowledge. These techniques of knowledge acquisition were used in an extension of ALICE known as 'AZ-ALICE'.

There are many variants in the implementation of AIML. The two major approaches are Program-AB (latest) and Program-O. Moreover, we can combine AIML with Android system as well so that a mobile based chat assistant can be deployed.

Chapter 3: Project Management Plan

In this chapter, we describe the entire plan which we followed for managing the project since its start till its completion. We first discuss the feasibility analysis which was performed to see if the project was feasible given the requirements for the project. There we discuss feasibility of the project with respect to various factors like time, budget, etc. Then, we discuss about the project life cycle where we see different phases of the project. Then, we discuss the cost and time estimation for the project, which is followed by the resource plan document, which gives the description about the resources we used. Then, we show how the tasks were distributed among the team members - who was assigned what responsibility and so on. Then, finally, we show the project timeline describing each and every phase of the project in a timeline.

3.1 Feasibility Analysis

In this section, we discuss various feasibility studies we performed which includes economical feasibility, operational feasibility, technical feasibility, legal feasibility and time feasibility.

3.1.1 Economical Feasibility

The software application requires tools / plugins which are open source, thus there is negligible cost associated with the tools or plugins. So, overall the software application is economically feasible.

3.1.2 Operational Feasibility

The application will require Java installed on the PC for execution. For the application based on second approach, the software application requires Java Runtime Environment (JRE) for launching the chatbot applet and is completely functional on a website. Since, Java is easily available and installable, and it provides good platform independence, the project is operationally feasible.

3.1.3 Technical Feasibility

The project is technically feasible as the technologies which are required to develop the application (Java, IDE, API, and other additional softwares) are available and as discussed before, at a negligible price. Also, the software is dependent on the Internet for handling

communication with the bank's databases and since, the Internet is available almost everywhere, it doesn't pose any risk on the feasibility.

3.1.4 Legal Feasibility

The software is legally feasible. All the tools used, APIs used and additional resources used are legal and verified and there are certainly no copyright violations. Also, the project follows the norms defined by the institution regarding the definition, development, distribution and evaluation of the project.

3.1.5 Time Feasibility

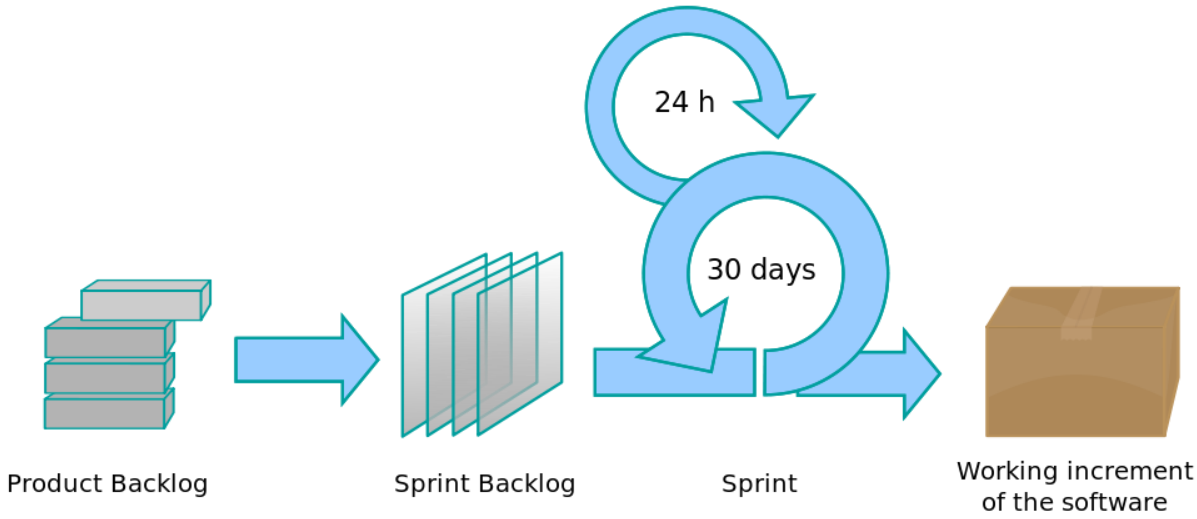
The project is somewhat feasible with respect to time. The schedule which was defined for each phases of the project and the completion of the project was appropriate. Also, the time allotted for completion and submission of the project by the institution was appropriate.

3.1.6 Conclusion of Feasibility Study

Thus, we have successfully completed the feasibility study. We conclude that the project is feasible with respect to money, operations, technologies, legal norms and also time.

3.2 Life cycle model

We follow the Scrum Agile development method. The life cycle for the development is shown in the figure below.



We followed Scrum Agile development because, it allows defining of current backlog (or list of tasks to be done) and then rigorously work on some items of the backlog (sprint) and deliver the increment as soon as possible. This allowed us to develop the project module by module and also refining the backlog as the requirements change or when the set of tasks are completed. At the end of each sprint, we get a working part of software. This way, incrementally, we can rapidly work on a set of tasks, implement them, and add to the software.

3.3 Project cost and time estimation

The external aids used in development and implementation of this proposed project are Open source, hence the project cost is negligible. The time estimated for completion of the project is 10-12 months approximately.

3.4 Resource Plan

As there are three members in the team the deliverables achieved perfectly co-ordinate and are communicated simultaneously. The proposed system is developed using two different approaches.

The resources used along with their usage in different modules:

Modules	**First Approach**	**Second Approach**
Development methodology	Agile - SCRUM	
Language used	Java	AIML
User interface	Java Applet	Web technologies: (html,css,php) & Command line
IDE or Editor used	NetBeans IDE	Sublime Text 3

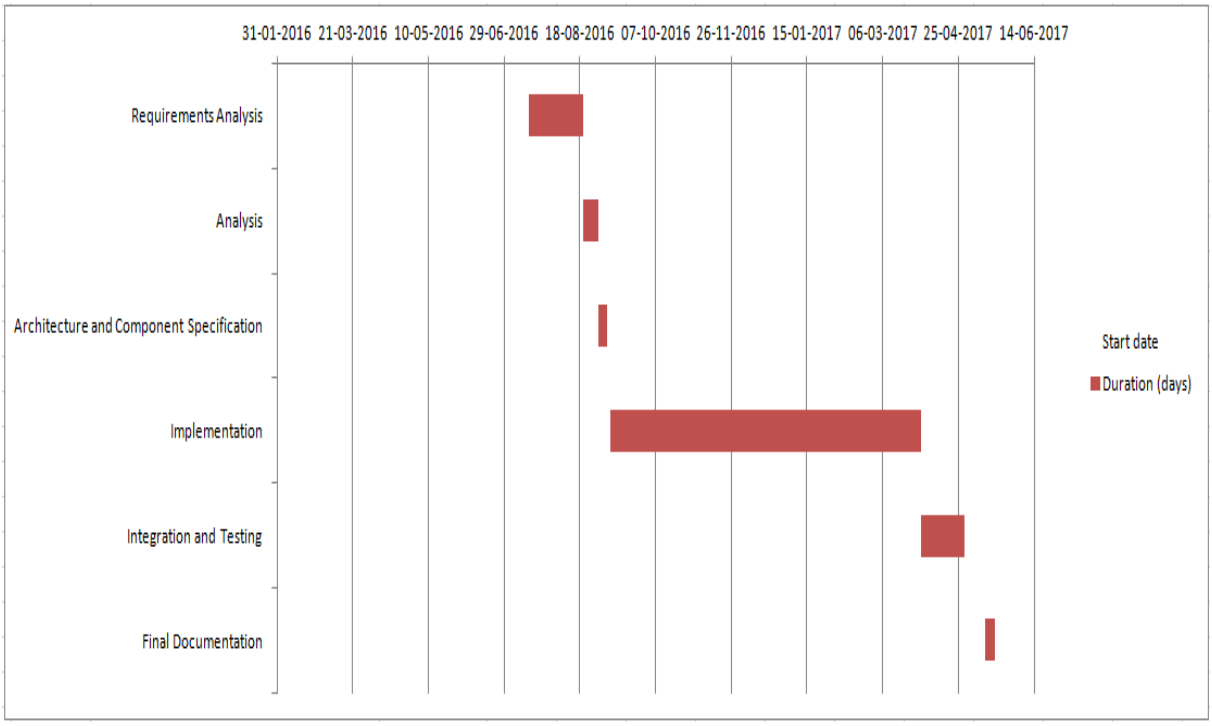
Database tools	JDBC	XAMPP Server , Excel
Testing	Black Box	
External aids	Stanford Parser	Pandorabots from ALICE Foundation

3.5 Task and Responsibility Assignment Matrix

Tasks	**Kiner B. Shah**	**Darshan P. Shah**	**Mohit S. Shetty**
Research	Yes	Yes	Yes
Text Normalization	Yes	Yes	Yes
POS Tagging	Yes	Yes	No
Phrase structure Parsing	Yes	No	Yes
Dependency Parsing	Yes	No	No
Dependency to Relations rules formation	Yes	No	No
Relation matching	Yes	No	No
Pattern matching	No	Yes	Yes
Data gathering	Yes	Yes	Yes
UI	Yes	Yes	Yes
Testing	Yes	Yes	Yes

Documentation	Yes	Yes	Yes
Project competition and paper publishing	Yes	Yes	Yes
AIML research	Yes	Yes	Yes
AIML dataset and knowledge creation	No	Yes	Yes
AIML coding	No	Yes	Yes
Exploring the app for Android system	No	Yes	No

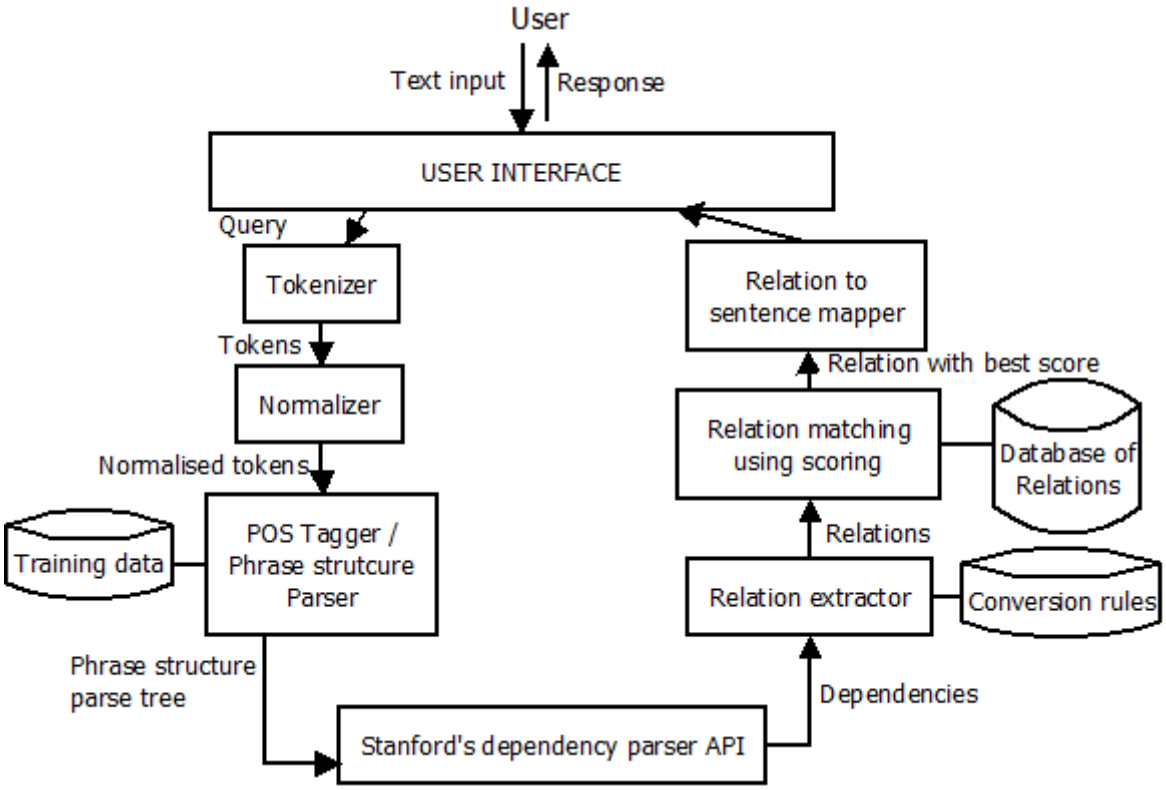
3.6 Project Timeline Chart

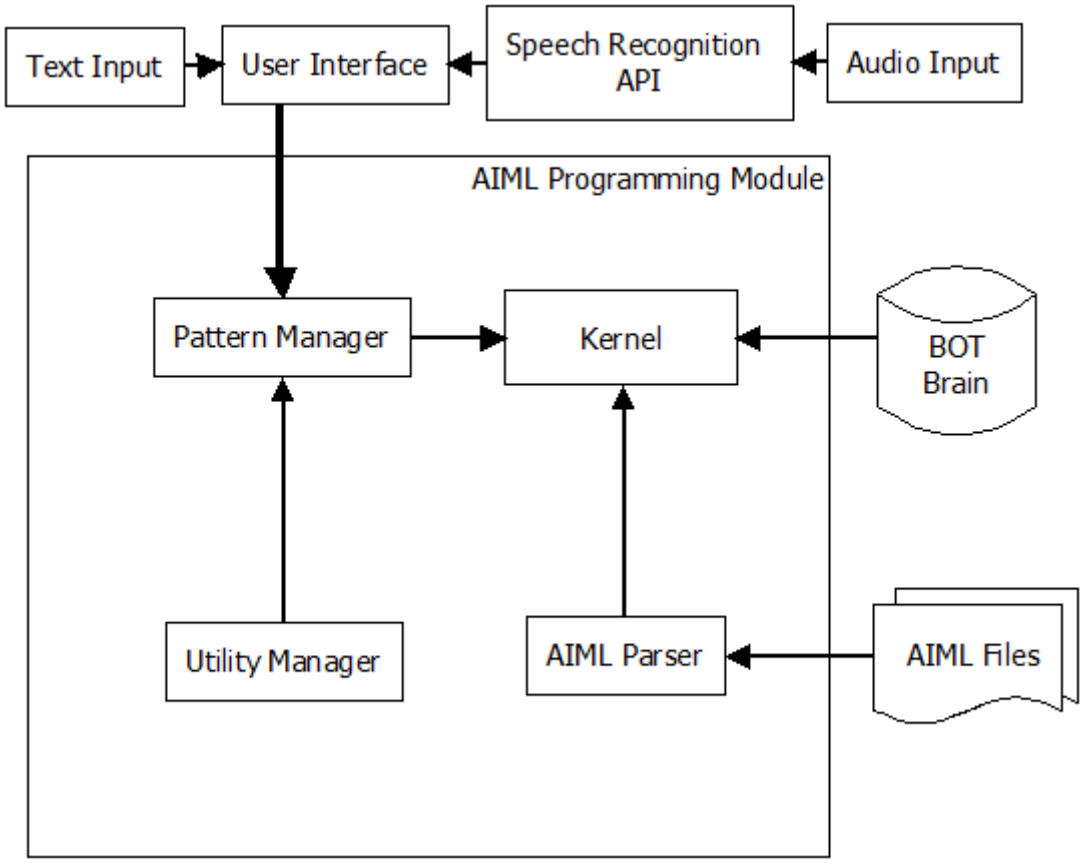


Chapter 4: Project Analysis and Design

4.1 Software Architecture Diagram

The software architecture diagrams for the two approaches used are shown below:





4.2 Architectural style and justification

4.2.1 Chosen System Architecture

First approach:

1. User Interface module: A user interface is provided to the users which contains text field for users to type their queries, a text area for providing response, and various screen-based controls for different operations.
2. Tokenizer module: Converts sentence into series of tokens or words.
3. Normalizer module: Normalizes the tokens - spell correction, conversion to lowercase, and acronym expansion.
4. POS Tagger - An optional module. Assigns labels to the words which signify their semantic category.
5. Phrase Structure Parser: Generates a parse tree for the input sentence based on a pre-defined grammar.
6. Dependency Parser: Converts phrase structure parse tree to dependencies tree which contains various dependencies between different words in the sentence.
7. Relation Extractor: Converts obtained dependencies into relations which can be used for matching and finding response with respect to a relation database.
8. Relation Matcher: Finds the best relation out of the database that may answer the input query. It includes matching of the input relation and the relations in the database and finding the relation based on highest score.
9. Relation to sentence mapper: Maps the final relation to its corresponding sentence in the database and returns it to the user via the GUI.

Second approach:

1. The two main modules we need to consider here are Pattern Manager and AIML Parser.
2. This architecture follows a Data-driven Pattern matching approach which requires all the general queries to be already present in the AIML files along with their solutions in advance. Admin can add, improve or delete a query directly from the Command-line interface using certain predefined AIML tags , explained in section 5.1.2.
3. Pattern Manager: As the query is asked, the pattern manager looks whether that particular query in the AIML files or not.

4. AIML Parser: It's the actual representation of an AIML code. It also helps in categorizing the different types of queries. e.g. if you want to teach your chatbot about science than upload science.aiml and your chatbot can then use the responses stored in the database to answer question related to science to end users.
5. Bot Brain: It can answer any "What is" question by looking up the answer on the Internet i.e. the once which are not present in the AIML file database.
6. Kernel: It is responsible for all processing. It requires other classes for processing to happen, it require AIML parser, word substitution, utility manager and pattern manager for matching. It include following modules like sessions, bot predicates, word substitute, element processor. It also include brain file. If brain file is provided kernel tries to load the brain and if brain file is not their then it attempts to load all aiml files.

4.2.2 Discussion of alternative Design

The current architecture is not a tiered architecture. To support users from large geographical regions, it's important to have tiers which will help in distributing the load of all processing on a single computer. Two-tier architecture will help in distributing the application logic and databases on a server and keeping only front-end UI on the client computer. Three-tier architecture takes this to next level by introducing a new tier and distributing databases on that tier thus separating the application logic tier. There are many ways of organizing the distribution in three-tiered architectures.

4.3 Software Requirements Specification Document

4.3.1 Overall Description

4.3.1.1 Product Perspective

The product is a replacement to the existing systems. The existing systems sometimes lead to loss of efficiency in completion of tasks. This product is aimed at increasing the efficiency of tasks, thus saving the time spent for it by effectively utilizing the available technologies.

4.3.1.2 Product Functions

The product will do following main functions:

1. Communication of the users with the system by asking queries in the form of text (or speech).

2. Basic banking tasks like applying for new cheque books, viewing balance sheets, etc.
3. Authentication of users.
4. Processing with user's queries and providing response in text and /or speech form.

4.3.1.3 User Classes and Characteristics

User class	**Characteristics**
Bank customers	May have some knowledge about using the computer and browsing through the Internet. Need not be skilled users.
Developers	Must be expert users and their role is to maintain and enhance the software after being deployed.
Database Administrators	Must be expert users and their role is to manage the bank databases, defining access privileges for the bank staff.

4.3.1.4 Operating Environment

In general, the systems will require minimum of 2GB of RAM and 2GB of hard disk space.

The application, based on first approach, can be run on any platform provided Java is installed on that system and there is appropriate Internet connectivity.

The application, based on second approach, is web-based and works on a windows platform, precisely windows XP and higher. Chatbot applet works on an instance of Java Runtime Environment - Java plugin software is required.

4.3.1.5 Design and Implementation Constraints

As very few systems are deployed for the same application in the past, it was difficult to start from the basic prototyping, selecting the approach/model for implementation. Also, the training dataset was not readily available for use.

It was difficult to understand the mental model of the system from users perspective and what users expect from the system.

The choice of implementation language was also difficult to make due to varying skill sets of the developers and limited number of APIs available in the implementation language, if required to be used, as it could result in less complexity during integration.

4.3.1.6 User Documentation

Appropriate documentation will be provided to users so as to enhance their experience with the interface and ensure faster learning. This documentation includes,

1. User manuals
2. Reference help

See Appendix 2 and 3 for more information.

4.3.1.7 Assumptions and Dependencies

There are some assumptions and dependencies with respect to our project. We have assumed that the target platforms will have latest version of Java installed (Java 8). The dependencies associated with our project are:

1. The project is dependent on the APIs: Stanford Dependency Parser, Java Mail API, Driver for JDBC, API for Text-to-speech.
2. The project is dependent on MySQL which is the underlying database technology and which is hosted by Xampp server.

4.3.2 External Interface Requirements

4.3.2.1 User Interfaces

To make the system more simple and attractive, software tools like available bootstraps can be used. *Bootstraps* are readily available modules that can be integrated with Web Technologies like HTML for the enhancement of the User Interface. We have also tried launching the bot on pandora bots playground. The user interfaces are designed using Java Swing and using HTML / CSS. The user interfaces will provide a graphical interface to the users so that they can access the services more easily. It will contain login and registration forms, input field for queries, help section, etc.

4.3.2.2 Hardware interfaces

Our project will have some hardware interfaces, like microphone for giving voice commands and speakers / headphone for listening to voice output.

4.3.2.3 Software Interfaces

Software interfaces include, various APIs used like Stanford Dependency Parser API, Java Mail API, JDBC driver, API for Text-to-speech, and also, Xampp server.

We have used the Program-O interface which has inbuilt UI and backend database processing capabilities.

4.3.2.4 Communication Interfaces

User requires a headphone /microphone in case of Voice-enables communication. This software is available as an android application providing a touch interface.

4.3.3 System Features

4.3.3.1 Customer Interaction

The main goal of the system is to allow customers to interact with the system, ask queries, and get some response from the system which may be either response to an inquiry or execution of a command given by the customer.

4.3.3.2 Voice-enabled Chat

Inclusion of an anthropomorphic method in software makes it more desirable to use. It saves time and sometimes users feel more comfortable to use software which is more human-like. Also, this gives users flexibility especially when they can't type the queries (like when their hands are occupied).

4.3.3.3 Basic bank - related services

It includes banking activities such as applying for new cheque books, viewing balance sheets, etc. This reduces paper works and gradually increases use of technology in a smart way.

4.3.3.4 Security and Privacy

Security is provided to the 'bank account holders' by using appropriate 2-factor authentication in which One Time Password (OTP) along with password is used. Also, check for actual

account holders is done by verifying the hashes obtained by their input values (during registration for application) and the values they provided during the opening of their accounts. The accesses given to the database can be limited which ensures that only the authorized users make changes to the databases which are usually database admins. Admin, however, may assign appropriate access rights to the bank staff.

4.3.4 Other Non-functional Requirements

4.3.4.1 Performance Requirements

The software application can work on a 32/64 bit Windows XP or higher with minimum 2GB RAM. Good Internet connectivity is required.

The processing capability of the machine and the storage should be high. Preferable are i7 and 1TB hard disk so that database fetching operations are done faster and queries are resolved faster.

4.3.4.2 Security Requirements

The application using first approach is a desktop application. Appropriate security measures must be taken to reduce the risk of privacy breaches. Authentication, integrity check mechanisms and encryption should use latest algorithms (which are not broken). For database servers, appropriate multi-level security must be implemented.

Since, the application (based on second approach) is a web based application, it has to be seen that the browser should be robust to cyber attacks. Browsers which support latest or recent versions of TLS must be used like Google Chrome. A verified antivirus and anti-malware software has to be installed and if already installed, must be kept up to date. Besides this, a firewall must be used to block malicious content and if already used, its policies needs to be reviewed and updated if required.

4.3.4.3 Software Quality Attributes

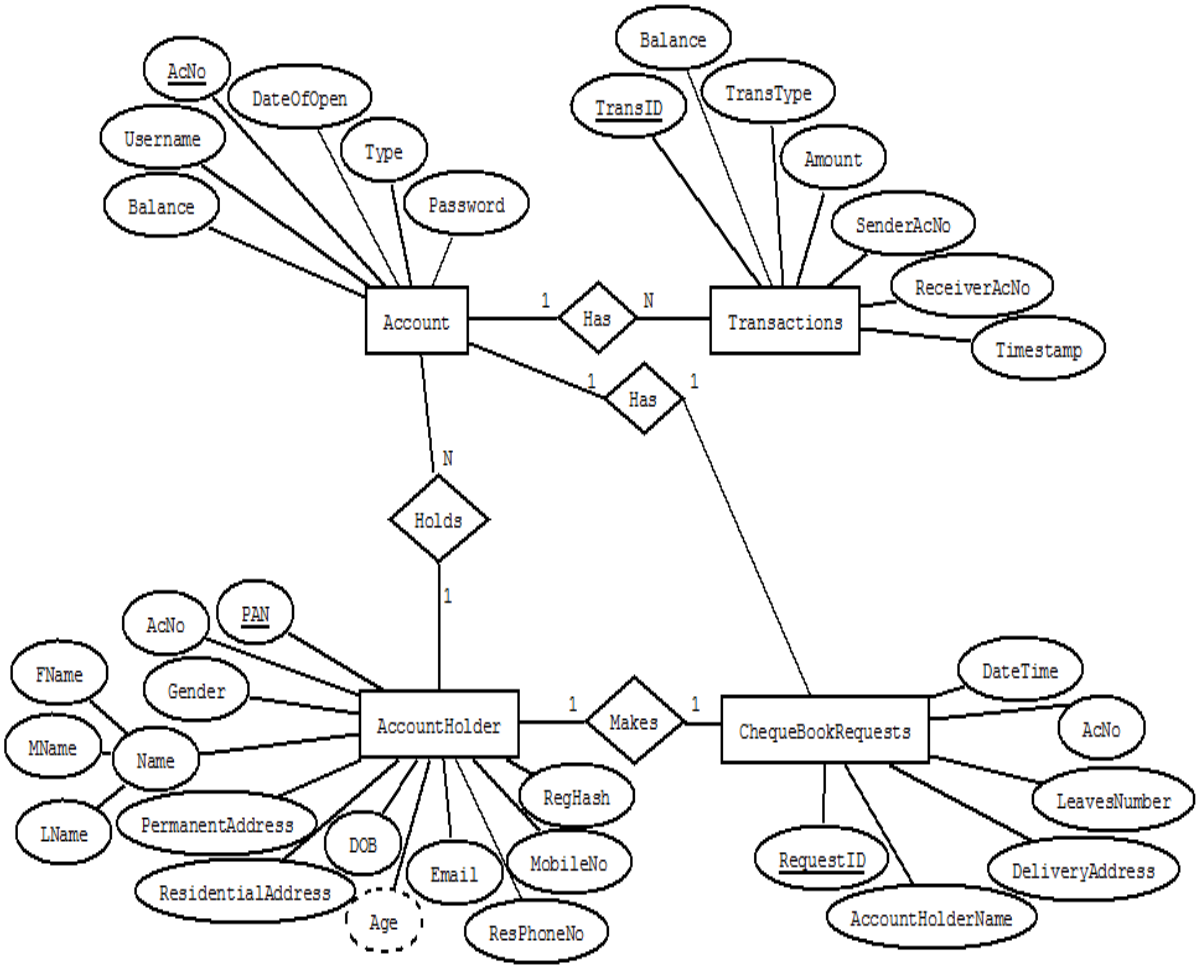
Category	**Quality Attribute**	**Description**
Design	Conceptual	For the current proposed system, the database is

Qualities	Integrity	centralized and not distributed. Hence, all the changes that are made in the system are reflected on this backend database following strict consistency and ensuring integrity.
	Maintainability	The system can be maintained without much expertise by keeping up-to-date with the Banking and Financial information for prolonged usage of proposed software.
	Reusability	The system is very flexible to modifications and can be reusable. Modifications can be increasing the Dataset or adding a different functional specific chatbot. For eg.: Instead of Banking chatbot ,we can create a chatbot for just Loan purposes by making slight modification in the Data set.
Run-time Qualities	Availability	The application will be available 24 by 7.
	Interoperability	Each module is functional independent. Only Data sharing is required between different modules for filtering (normalization, tagging, Parsing) , comparing, recognition and returning back the response to user interface.
	Manageability	The most important thing to manage is User chat history and Account holder's Data which is managed using an appropriate Database design.
	Performance	System gives appreciable performance however can be expected to give even higher performance after sufficient optimization.
	Reliability	A replicated set of banking dataset is kept and updated after certain interval of time. This will ensure that the system will be reliable if one site fails and we will

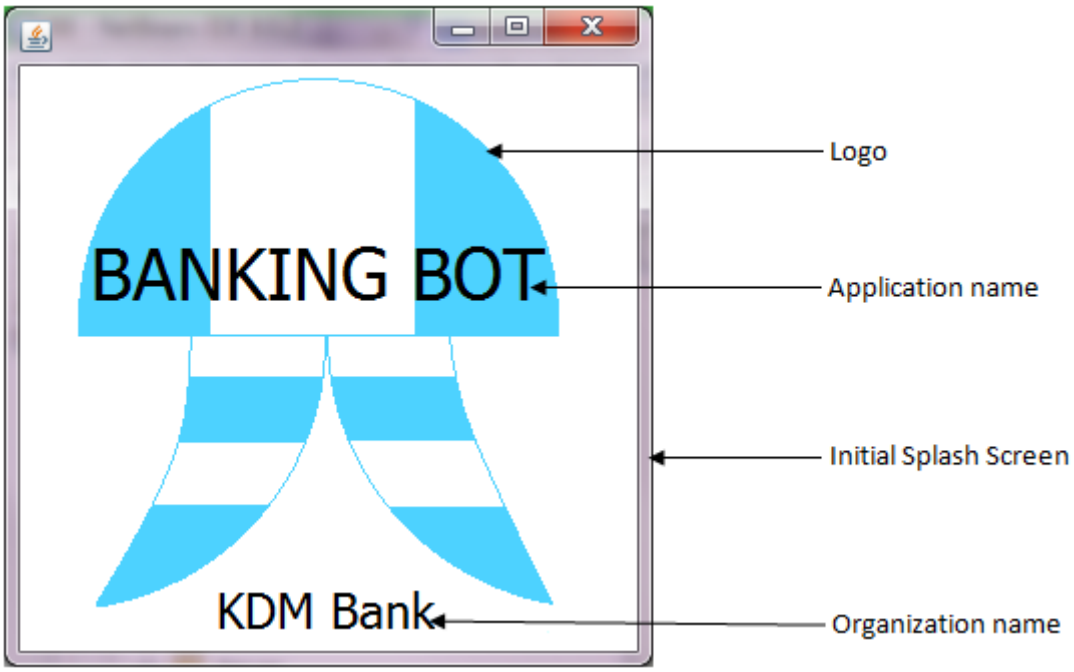
		resume the system.
	Scalability	This application can be extended for any number of customers and there are no limitations on the size of data as well.
	Security	This application doesn't require storing much of confidential data. Only Bank account holders are required to authenticate.
System Qualities	Supportability	The proposed system has proper user context and technology/resource plan available. Even for updating or modifying the resources available are sufficient to meet the requirements.
	Testability	Testing is required in text and speech recognition modules i.e. whether the system corrects the user words appropriately and also recognizes the user's voice, or not.
User Qualities	Usability	The system offers a simple user interface and is function specific for better understanding and less confusion. Also the system corrects the wrongly spelled words and reduces user work thus making the application more desirable and usable.

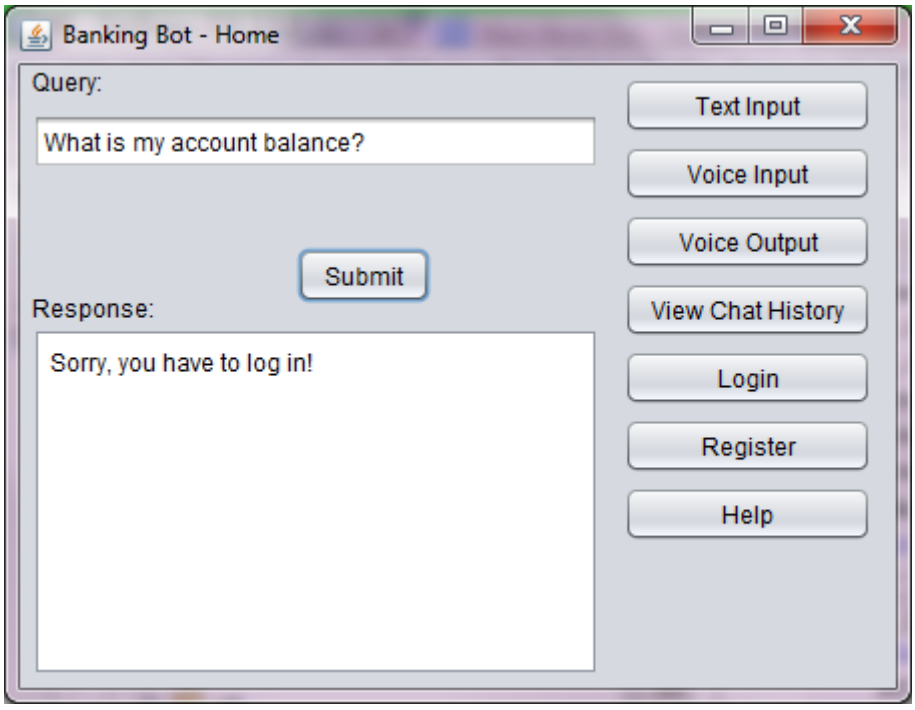
4.4 Software Design Document

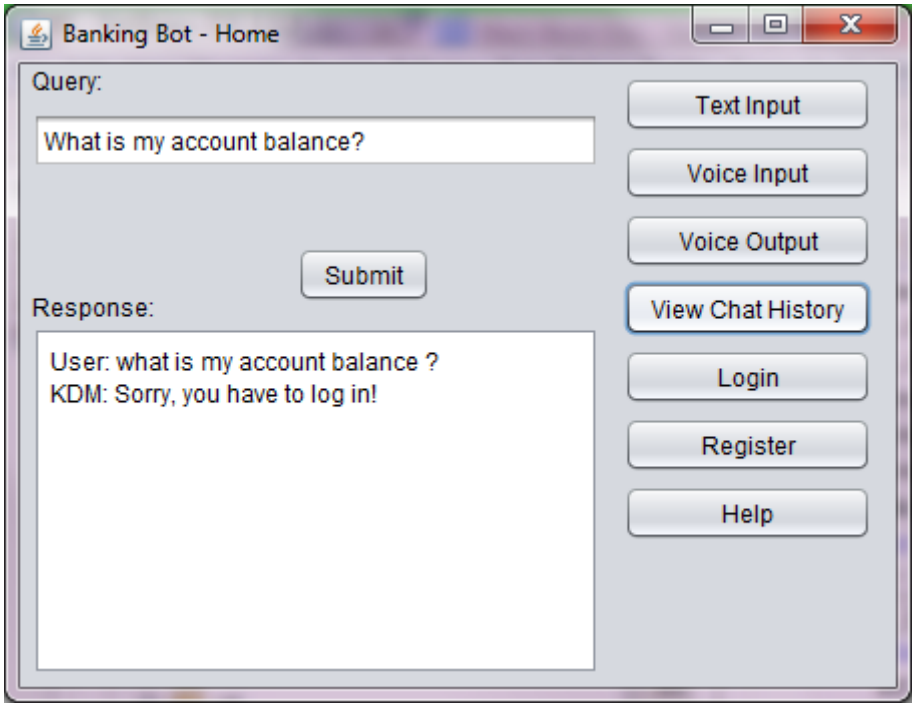
4.4.1 Database design (ER Diagram)

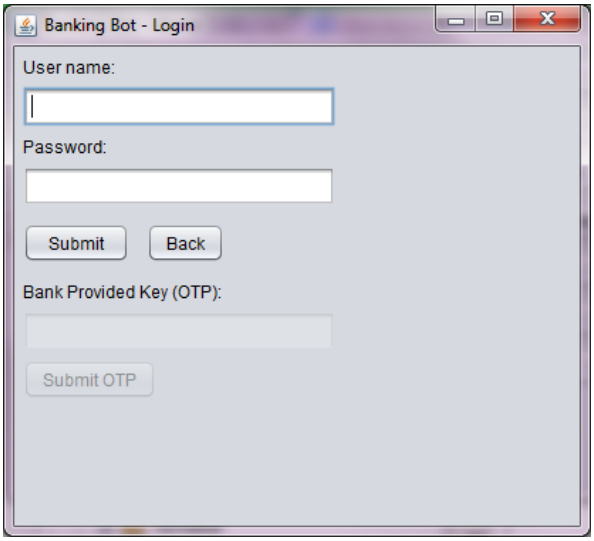


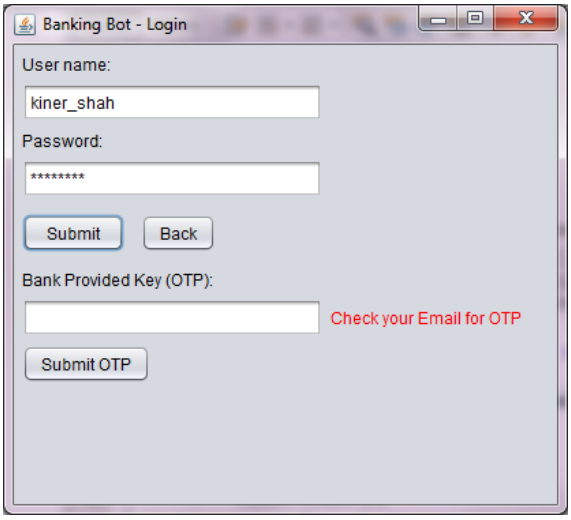
4.4.2 UI Design

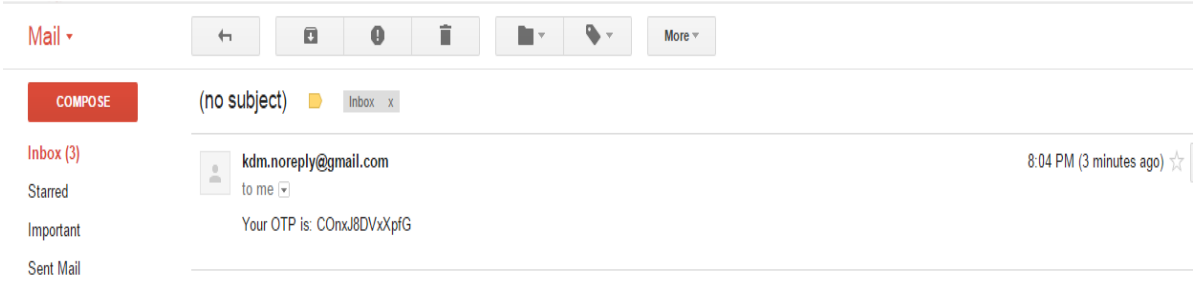


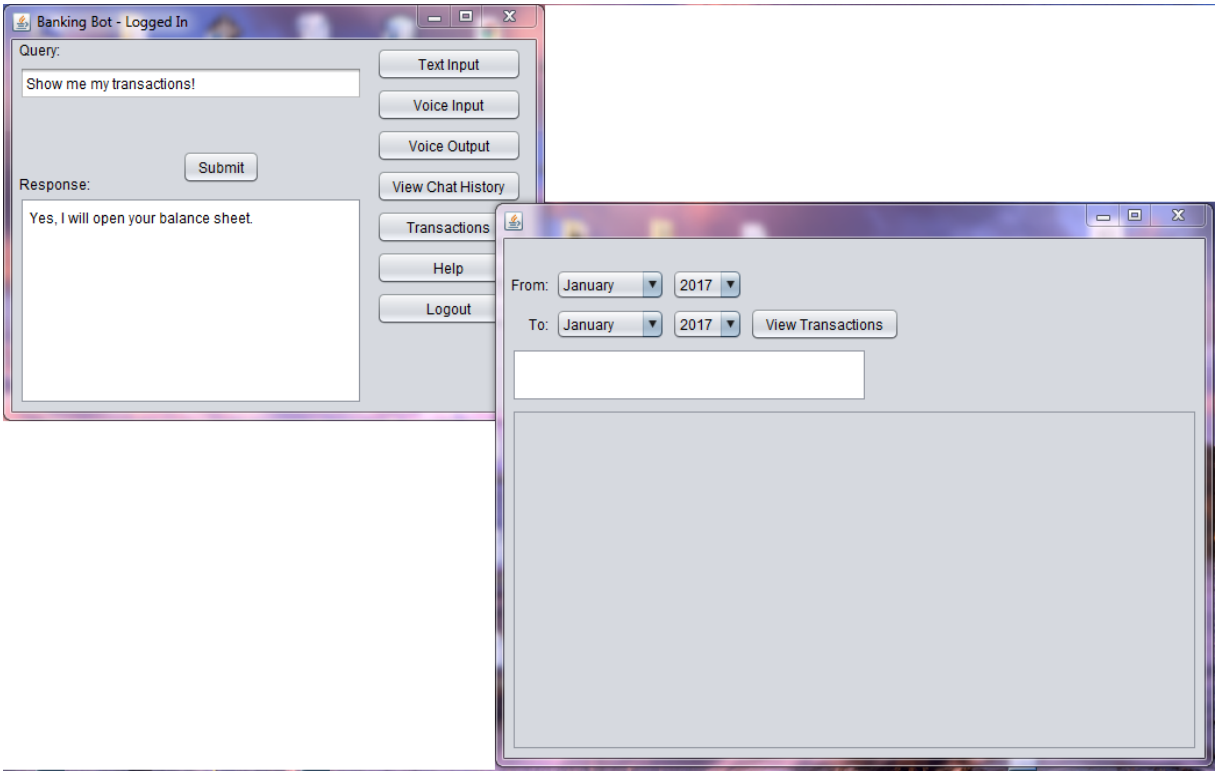


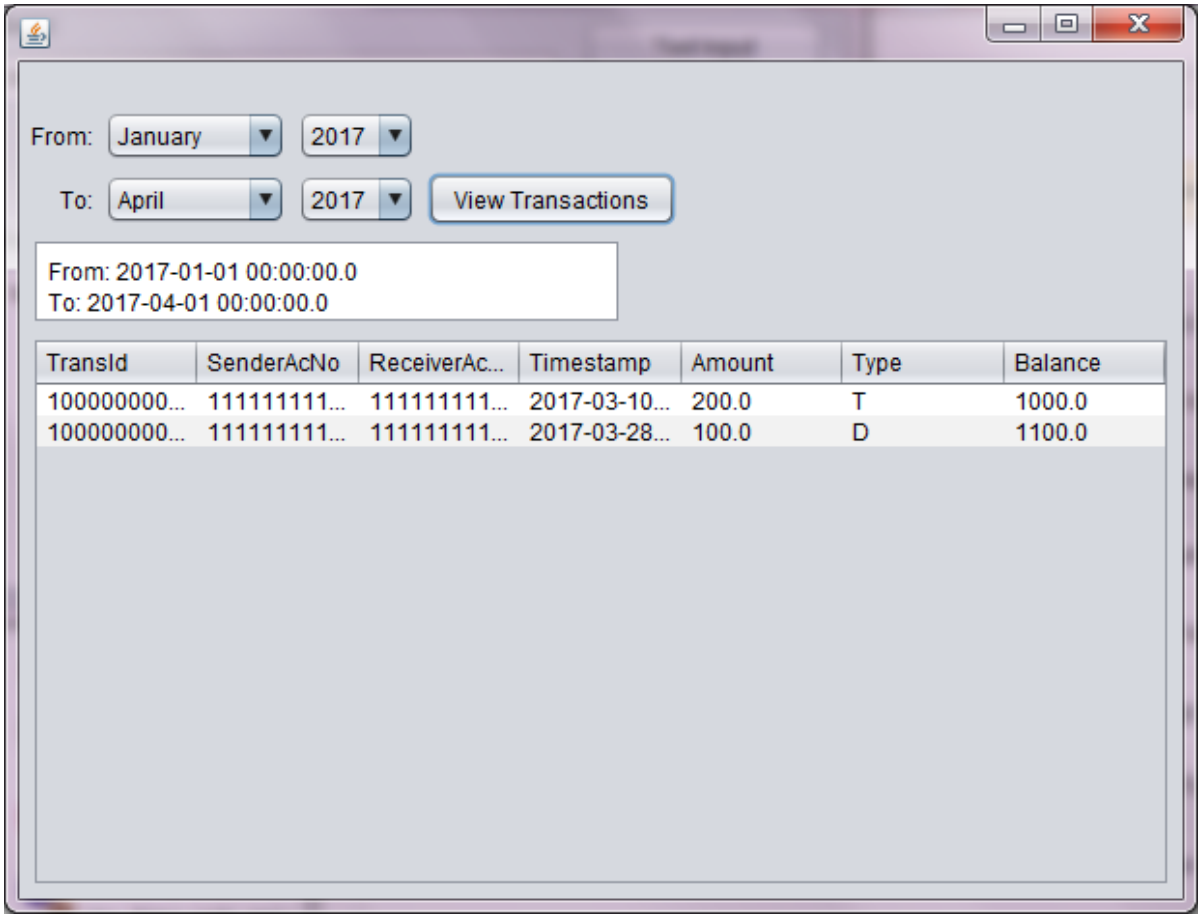

Banking Bot - Home
Query:
What is my account balance?
Submit
Response:
User: what is my account balance ?
KDM: Sorry, you have to log in!
Text Input
Voice Input
Voice Output
View Chat History
Login
Register
Help

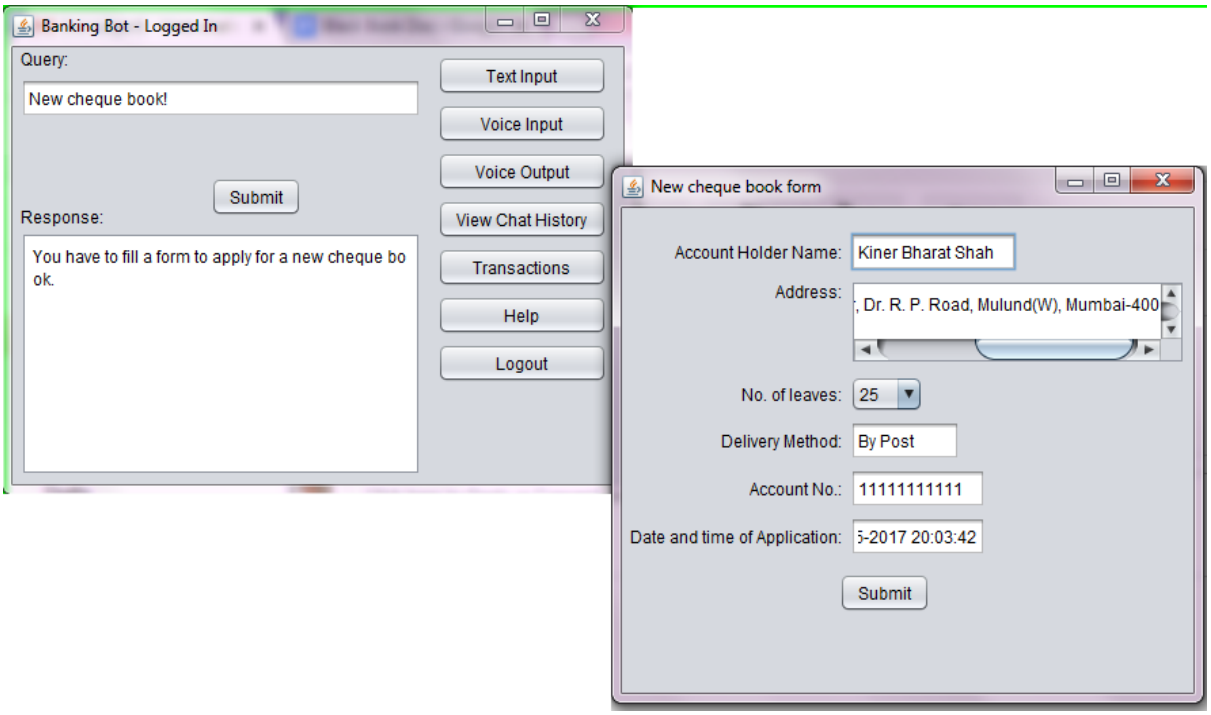

Banking Bot - Login
User name:
Password:
Submit
Back
Bank Provided Key (OTP):
Submit OTP

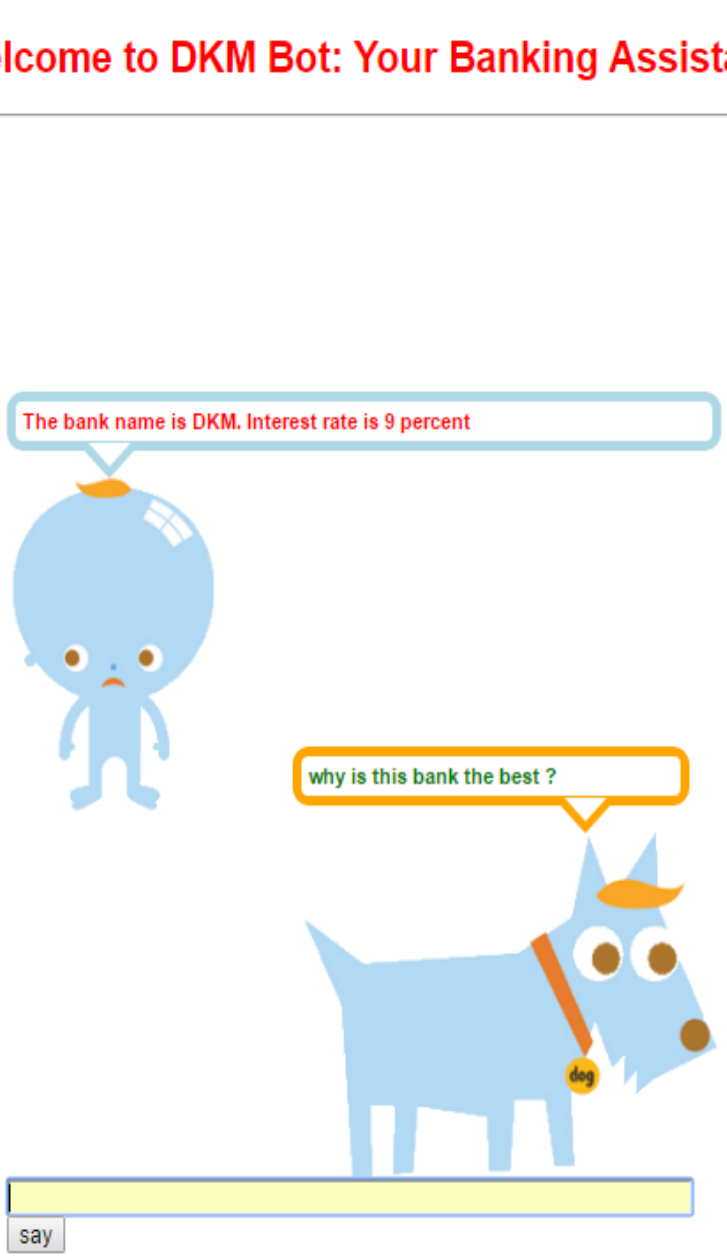

Banking Bot - Login
User name:
kiner_shah
Password:
****
Submit
Back
Bank Provided Key (OTP):
Check your Email for OTP
Submit OTP


Mail
More
COMPOSE
(no subject)
Inbox x
Inbox (3)
Starred
Important
Sent Mail
kdm.noreply@gmail.com
to me
8:04 PM (3 minutes ago)
Your OTP is: COnxJ8DVxXpfG

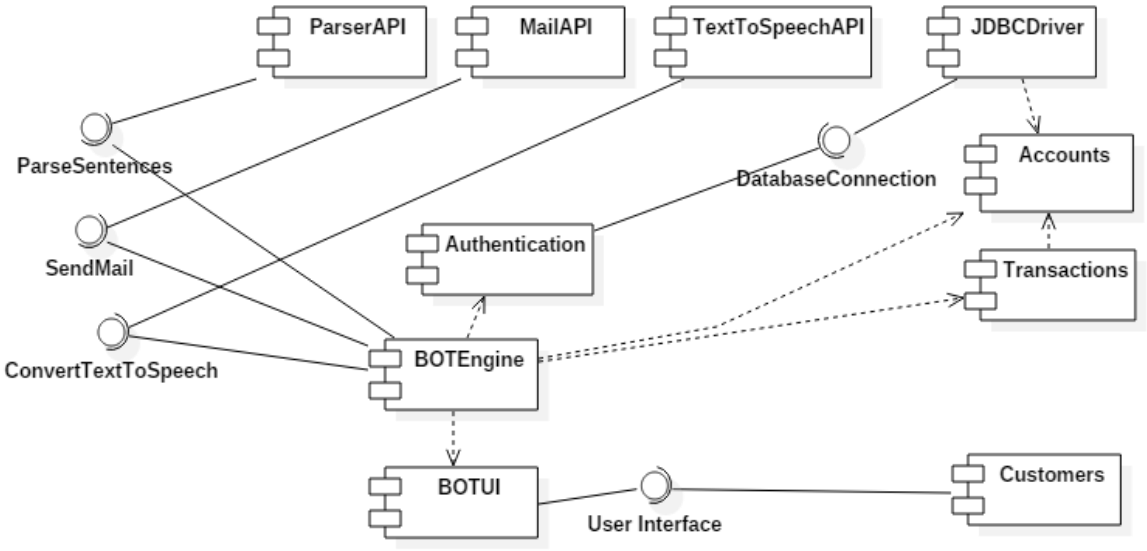

Banking Bot - Logged In
Query:
Show me my transactions!
Submit
Response:
Yes, I will open your balance sheet.
Text Input
Voice Input
Voice Output
View Chat History
Transactions
Help
Logout
From: January 2017
To: January 2017
View Transactions


From: January 2017
To: April 2017
View Transactions
From: 2017-01-01 00:00:00.0
To: 2017-04-01 00:00:00.0
TransId SenderAcNo ReceiverAc... Timestamp Amount Type Balance
100000000... 111111111... 111111111... 2017-03-10... 200.0 T 1000.0
100000000... 111111111... 111111111... 2017-03-28... 100.0 D 1100.0

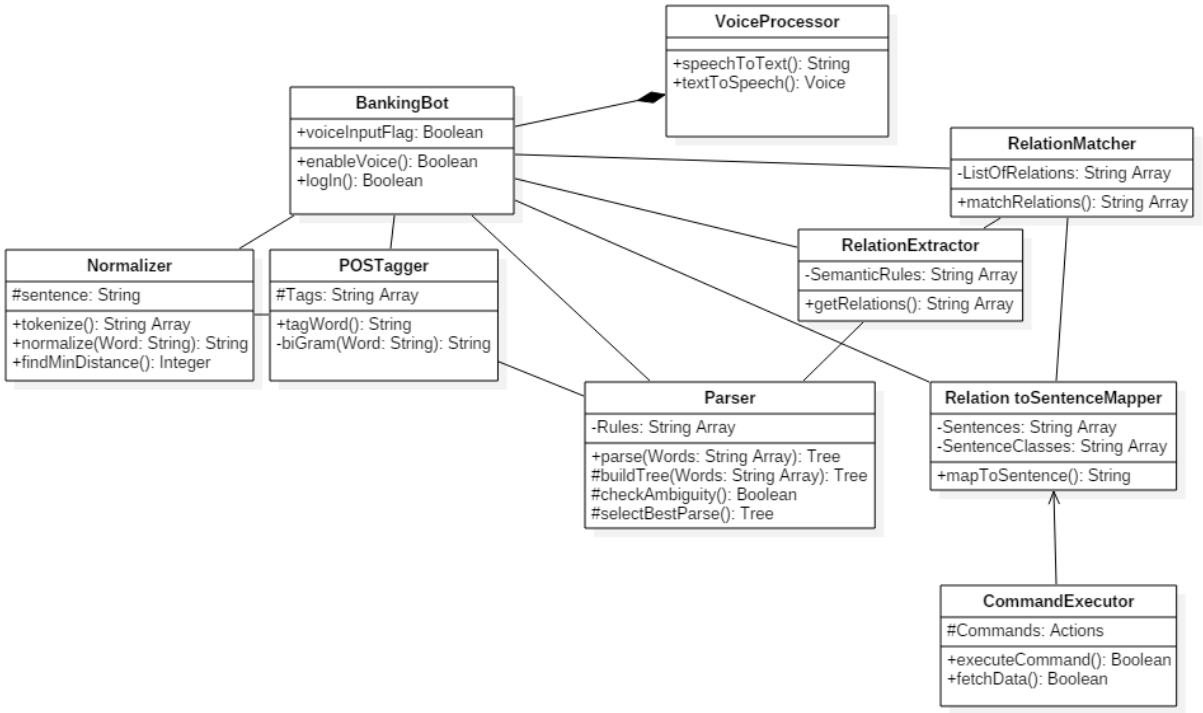

Banking Bot - Logged In
Query:
New cheque book!
Submit
Response:
You have to fill a form to apply for a new cheque bo ok.
Text Input
Voice Input
Voice Output
View Chat History
Transactions
Help
Logout
New cheque book form
Account Holder Name: Kiner Bharat Shah
Address: , Dr. R. P. Road, Mulund(W), Mumbai-400
No. of leaves: 25
Delivery Method: By Post
Account No.: 11111111111
Date and time of Application: 5-2017 20:03:42
Submit


Welcome to DKM Bot: Your Banking Assistant
The bank name is DKM. Interest rate is 9 percent
why is this bank the best ?
dog
say

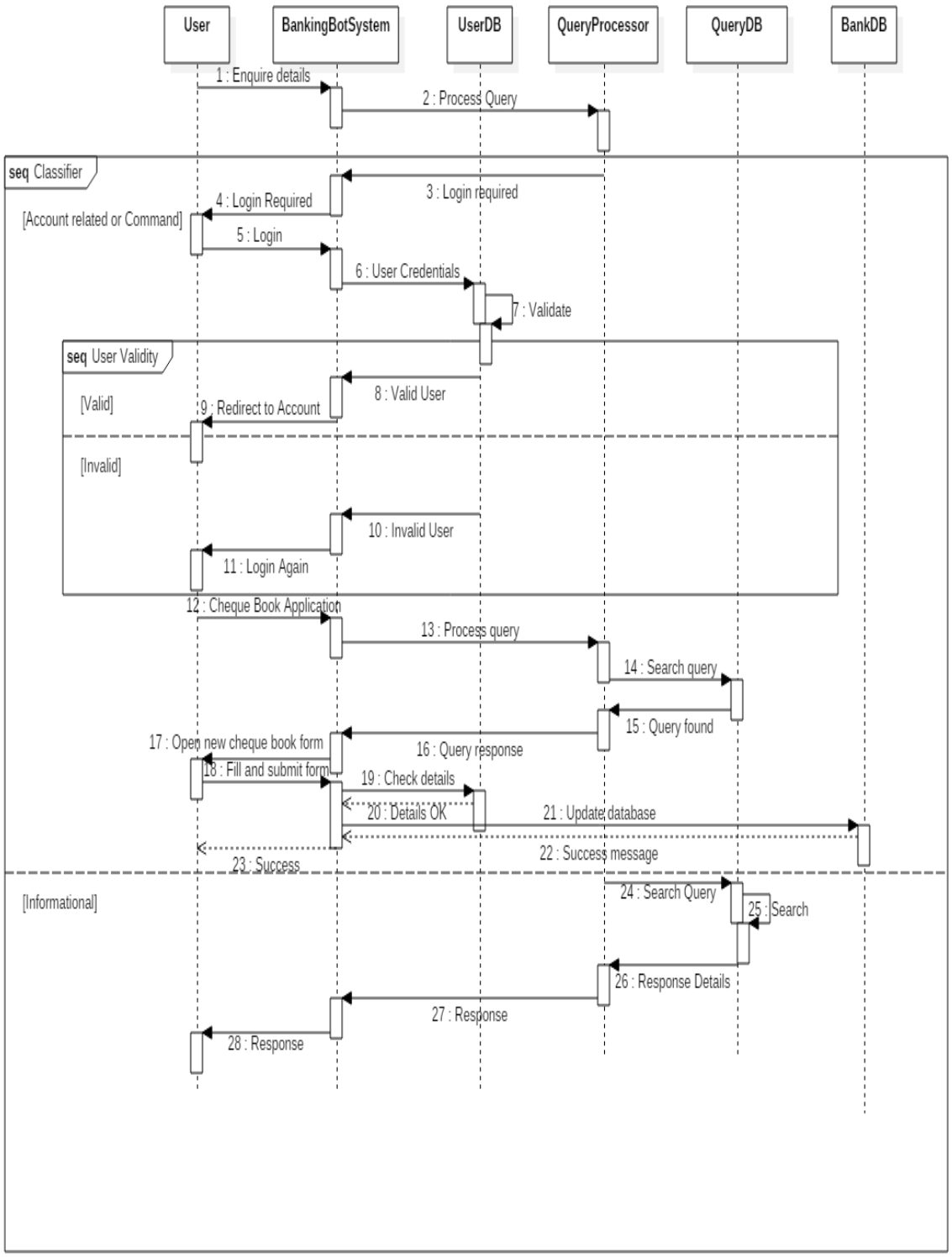
4.4.3 Component Diagram



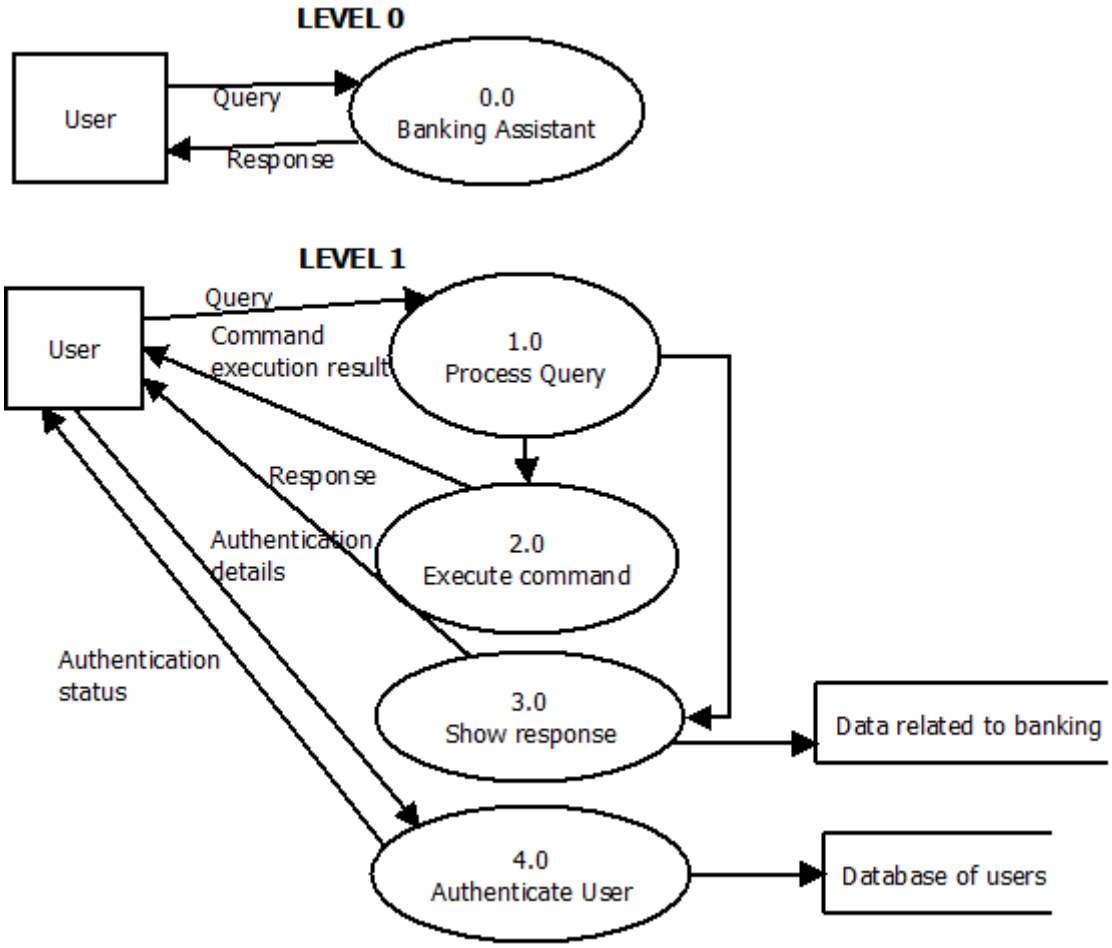
4.4.4 Class Diagram



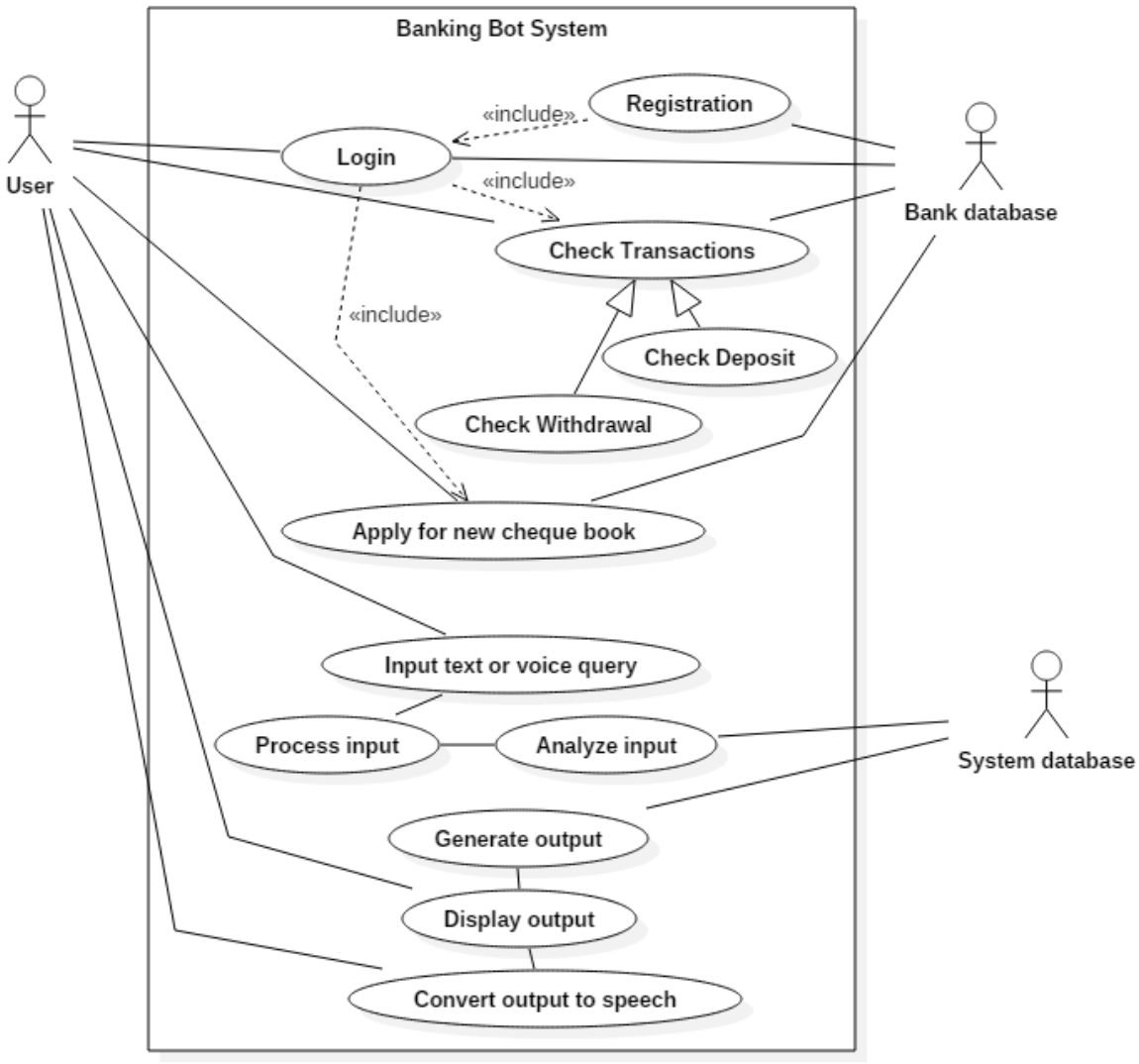
4.4.5 Sequence Diagram



4.4.6 Data Flow Diagram



4.4.7 Use Case Diagram



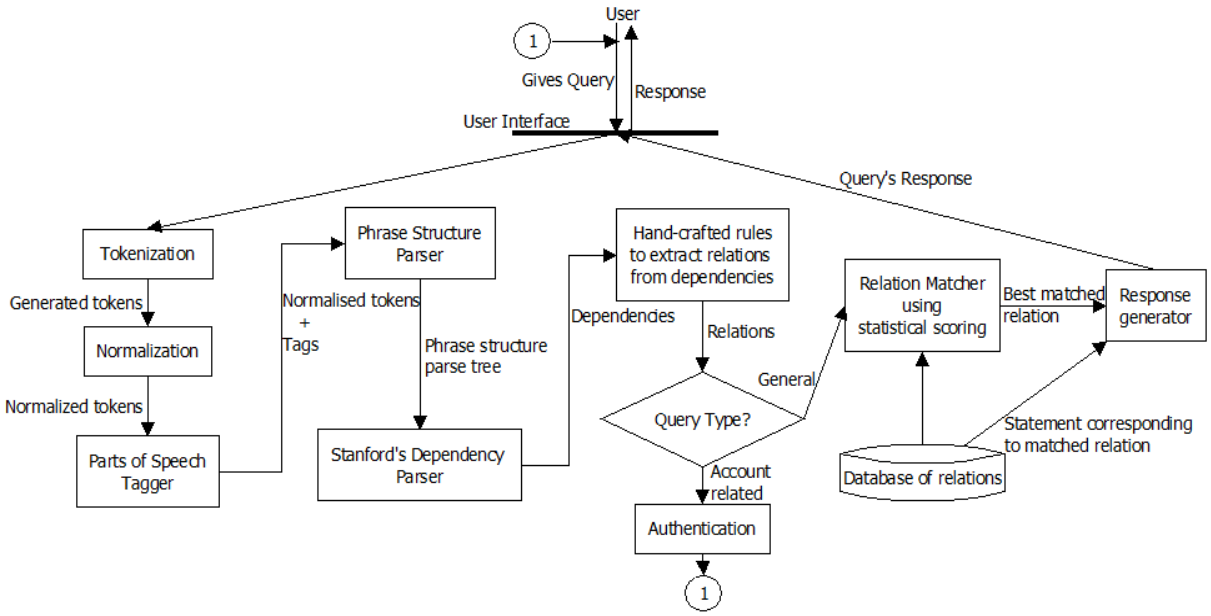
Chapter 5: Project Implementation

5.1 Approach / Architecture / Algorithm / Methodology

As mentioned before, we have implemented two approaches for building the banking assistant - first is using Natural Language Processing (NLP) and second is using AIML. In this section, we discuss both the approaches.

5.1.1 NLP-based approach

Natural Language Processing consists of techniques to process natural language text so as to use it effectively in various applications. Some of the common applications are speech recognition, information retrieval, question-answering, language translation, etc. We use NLP techniques for building the banking assistant. An assistant can be either an expert system or a question-answering system or a simple dialog-based system. The flow of the approach followed by us is shown in the below diagram.



Firstly, the user inputs his query via the user interface provided. The user interface forwards the query to the *Tokenization* module. The tokenization module is responsible for splitting the text into different tokens (or words) with respect to common delimiters like space or comma or semicolon. For example, if the user query is "*What is my account balance?*", the result of the tokenization module will be a set of words: {"*What*", "*is*", "*my*", "*account*", "*balance*", "*?*"}.

In the next phase, the tokenizer module passes the tokens obtained to the *Normalizer* module which normalizes the words into a standard format. It does the following tasks:

1. Conversion of words to lowercase
2. Removing of punctuation marks
3. Spell correction

Conversion of words to lowercase is done to make it easy for further processing. Removing of punctuation marks is done because punctuation marks usually don't have much significance in the processing. Spell correction, however, is an important task. It corrects some incorrectly inserted words which may cause problems during further processing, if not taken care of. We use Damerau-Levenshtein spell correction algorithm for this, which uses dynamic programming to compute dissimilarities between two words. For this, we use a list of correct words and try to match each input token with those words. If they match (i.e. they are 100% similar), then no correction is required. However, if they don't match (i.e. there are certain dissimilarities), then four operations are tried to make the words similar. The four operations are:

1. Insertion of a character (e.g. acount -> account)
2. Deletion of a character (e.g. acccount -> account)
3. Substitution of a character with some other character (e.g. accoumt -> account)
4. Transposition of two adjacent characters (e.g. accuont -> account)

Whichever operation leads to some correct word with the least cost, is chosen. But, this can sometimes still lead to erroneous results as more than one word may have the same least cost. For this, we use probability as the measure to select the word out of the ones which have least cost. The most probable word (i.e. the word with highest probability value) will be chosen. The Damerau-Levenshtein algorithm leads to $O(M. N. max (M, N))$ complexity in worst case. The output of normalizer is a set of normalized words.

Next phase, is *Parts of Speech Tagging*. This phase assigns a label to each word output by the normalizer. A label in this case is the class the word belongs to i.e. whether the word is a noun, or a verb or a preposition or an adjective, etc. This phase is optional as its following phase, Phrase Structure Parsing, does the same but with other approach. We implemented this phase with the aim that it may be explicitly used later. We used Hidden Markov Models (HMM) for this and implemented the HMM using Viterbi algorithm. A Hidden Markov Model is a probabilistic model which is used in various applications like speech processing. It uses Bayes theorem to predict which output state the input most likely belongs to. HMM defines three problems of which solutions are needed:

1. Given the observation sequence, $O = O_1, O_2, ..., O_T$ and a model, $\lambda = (A, B, \pi)$, compute the probability of the observation sequence, $Pr(O \mid \lambda)$.
2. Given the observation sequence $O = O_1, O_2, ..., O_T$, choose a state sequence, $I = i_1, i_2, ..., i_T$ which is optimal in some meaningful sense.
3. Adjust the model parameters, $\lambda = (A, B, \pi)$ to maximize $Pr(O \mid \lambda)$.

We try to solve the third problem in POS Tagging because we need to find a tag with maximum probability given the model.

Next phase, as mentioned above, is *Phrase Structure Parsing*. Parsing refers to the process of checking the syntax of a given language and see if a given input query follows the syntax. In case of Natural Language, grammar is not well defined because Natural Language is too vast and varies according to the language. However, we may some limited number of rules which most of the sentences may satisfy. We specify the rules using Context-free grammar (CFG). A CFG is a quadruple, *G = (V, Σ, R, S)* where *V* is a set of non-terminal symbols, *Σ* is a set of terminal symbols, *R* is a set of rules or productions, *S* is the start variable. But, there can be thousands of such rules and forming a parse tree for a given query will often lead to ambiguous results. So, to avoid any ambiguities, statistical scoring of the rules can be done by using probability as the score corresponding to a rule. Thus, the grammar is modified to include one more variable *P* which represents the set of probabilities for each rule. The grammar now becomes, *G = (V, Σ, R, S, P)*.

Let's take the previous example, "*What is my account balance?*", the phrase structure parse will result in the following parse tree:

```
(ROOT
 (SBARQ
  (WHNP (WP What))
  (SQ (VBZ is)
   (NP (PRP$ my) (NN account) (NN balance)))
  (. ?)))
```

In the above example, *ROOT* represents the root of the parse tree, *SBARQ* represents the start of a (question) sentence, *NP* represents a noun phrase, *NN* represents a noun, and so on. For phrase structure parsing, we use Probabilistic version of Cocke-Younger-Kasami algorithm,

which is a bottom-up dynamic programming algorithm, with worst case complexity equal to $O(n^3.|G|)$ where *n* is the length of the input query and |*G*| is the size of the grammar.

In the next phase, we convert the phrase structure parse tree obtained into *dependencies.* This step is important because phrase structure tree only tells that whether the input query is syntactically correct. To extract some semantic information from the queries, dependency parse tree is helpful. Conversion of phrase structure parse tree to dependencies is possible by using *Stanford Dependency Parser API*. Dependencies signify the relationship between different words in a sentence and thus are useful in providing semantic information.

Let's take the previous example, "*What is my account balance?*", the dependencies obtained after converting the phrase structure rules are shown below:

root(ROOT-0, What-1)
cop(What-1, is-2)
nmod:poss(balance-5, my-3)
compound(balance-5, account-4)
nsubj(What-1, balance-5)

The above output is of form *dependency_relation(word-index, word-index)*. For example, *nsubj(What-1, balance-5)* means the dependency relation is *nominal subject*, the dependent is *What* and dependee is *balance*. The numbers 1 and 5 represent the indices of the words in the sentence respectively. We obtain useful information from the dependencies. In the above dependencies, for example, we see that the *balance* is the subject of the sentence, *balance* is possessed by *me*, *account* and *balance* are treated together, etc. Since, Stanford Dependency Parser is not 100% accurate, there are certain cases where in different dependency labels are assigned to relations comprising of similar words. So, it will be unwise to directly use dependencies, and thus, some standard format is needed which will convert these dependencies into a form wherein similar relations labelled differently will be treated in the same way. By observation, rules can be formed which will convert dependencies to a standard format. For example, we can easily extract subject by not just *nsubj* relation but also, *xsubj*, *vocative*, and *nsubjpass* relations. Similarly, we can extract unique characteristics like color, category, etc. from *advmod*, *amod* and *dep* relations. There are many dependency relations

which will help in extracting some bits of information. This conversion is done by *Dependency to Rule conversion* module.

For example, if we take the above example, we will get the following relations

account-balance, Subject, null

account-balance, Possed_by, my

The relations are of format <*A, R, B*> where A is the dependent, R is the relation and B is the dependee. B = *null*, indicates the second argument is either not needed or is already used in other relation. Such relations are computed for each input query and the input relations are matched against a database of relations to find any similar relations which may be a part of the response.

Before going to the next phase, an important decision has to be made - how to ensure that other customers don't access any confidential data by firing queries. For this, the query has to be classified into either of the two classes that is, account-related or general queries. If account-related queries are asked and the customer isn't logged in the system, then the system must not execute that query and ask for customer's credentials as authentication. This is done by implementing a 2-factor authentication (using password and OTP in combination). The OTP feature is implemented by choosing a set of 10 to 15 alphanumeric characters (including uppercase and lowercase alphabets along with numbers) which are combined into one string and sent to the email address of the customer. For verification of correct OTP, SHA-512 hash is used - the hash of OTP is stored during generation and matched with the hash generated from OTP provided by customer.

The next phase, Relation Matching module, deals with this. Matching is done based on scoring as shown in the Algorithm below.

In the algorithm, the score to be added was set based on observations. The relation with highest score is chosen as the appropriate response relation. The sentence in the database corresponding to that relation will be given as output to the user. The output can also be given in the voice format. We have used FreeTTS API to convert the textual output in speech format.

1. Let $< A_i, R_i, B_i >$ be a set of relations for $i = 1$ *to* k in the database.
2. Let $< P_j, S_j, Q_j >$ be a set of input query relations for $j = 1$ *to* m.
3. Initialize *score* = 0.
4. *for* $i = 1$ *to* k
   for $j = 1$ *to* m
   if $A_i == P_j$ *and* $B_i == Q_j$ *and* $R_i == S_j$, *then*
   score = *score* + 10
   else if $A_i == P_j$ *and* $R_i == S_j$, *then*
   score = *score* + 5
   else if $B_i == Q_j$ *and* $R_i == S_j$, *then*
   score = *score* + 10
   else if $A_i == P_j$ *or* $B_i == Q_j$, *then*
   score = *score* + 2
   end for
   end for
5. return *score*
6. end

5.1.2 AIML-based approach

As mentioned above, we have also used the alternative approach for making the chat assistant called as AIML (Artificial Intelligence Markup Language). AIML is an extension of XML. AIML is basically used for pattern recognition of the queries from the knowledge database.

AIML is basically a rule based approach. Here we have our knowledge database in the templates stored in the AIML code. Whenever a user query is found, the patterns are checked from this knowledge dataset. If an answer is found, it will return the corresponding response or it will show - “No AIML category found”. Thus, we can say that it is just pattern matching and learning component is not present in AIML, but at the same time we can customize the bot in such a way using AIML tags that user isn’t aware that learning component is absent.

Some of the general tags that are used in AIML are shown in the table below:

Tag	**Description**

<aiml>	Defines the start of AIML.
<category>	Defines the unit of knowledge in knowledge base.
<pattern>	Defines the pattern to match what a user may input to an Alicebot.
<template>	Defines the corresponding reply to be given to the query.
<star>	Tag is used to match wild card * character(s) in <pattern> tag.
<srai>	Multipurpose tag, used to call/match the other categories.
<random>	Used <random> to get random responses.
	Used to represent multiple responses.
<set>	Used to set value in an AIML variable.
<get>	Used to get value stored in an AIML variable
<that>	Used in AIML to respond based on the context
<topic>	Used in AIML to store a context so that later conversation can be done based on that context
<think>	Used in AIML to store a variable without notifying the user.
<condition>	Similar to switch statements in programming language. It helps ALICE to respond to matching input.

One can add new set of Queries-Solutions for the bot to answer next time using 'Learn Tag'. The new set of Queries-Solutions can be made available while applying updates.(For eg: new Banking terminologies). One can improve Solutions for a Query using 'Prefer Tag'. This will be done after obtaining feedbacks from users to improvise the solutions.

5.2 Programming Language used for Implementation

Implementation highly depends on the programming language used and the type of that language - procedural or object-oriented, platform independent or platform dependent. Procedural languages like C, uses structured blocks to represent groups of information, for example, attributes related to account like account number, type of account, date of opening the account, balance, etc. must be stored together under one construct. Also, procedural languages are highly procedure dependent i.e. they work my means of functional constructs which are designed to carry out a single task. Object-oriented languages like Java, uses objects instead of structured blocks to store groups of information. These objects not only stores attributes but also functions which can be performed by accessing those objects.

Platform independence is an important characteristic as we don't want our application to run on machines wherein we first have to install additional software to run the system. These days, most of the developers try to build applications which don't require much prerequisites, however, in many cases, platform dependence is adopted, as certain software may be required to be installed in prior for correct working of application.

We use Java as the programming language for developing the assistant using the first approach i.e. using NLP. Java provides good platform independence and also, it is object-oriented. Also, various APIs are available in Java which makes integrating them easier.

There is another standard which is used for platform independence - XML. Extensible Markup Language (XML) is an extension to standard HTML, and allows creation of new tags and even new languages. There are many variants of XML, one of which was developed for supporting building of the assistants or so called chatterbots. Artificial Intelligence Markup Language (AIML) is a popular construct which was developed by Dr. Richard Wallace for writing a popular chatterbot, ALICE. We use AIML as the language for developing the assistant using the second approach.

5.3 Tools used

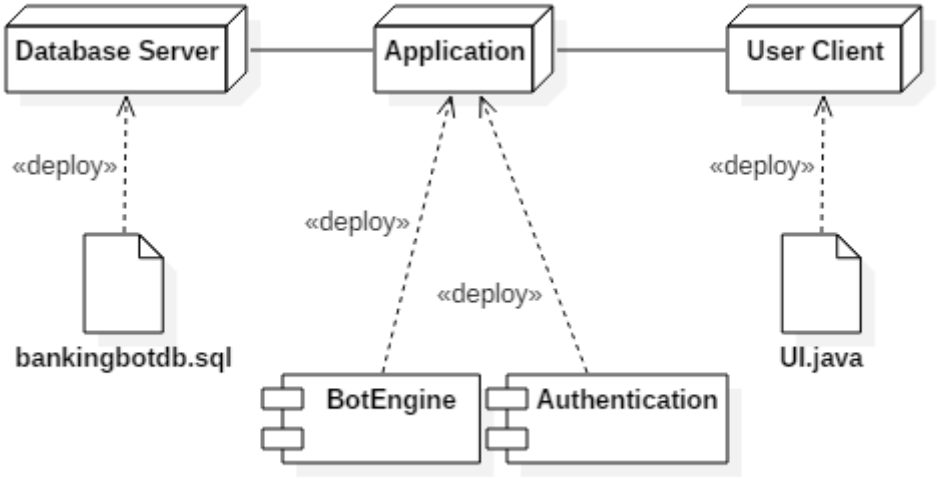
Tools are important for building applications. Without proper tools, application development can be very tedious. There are various classes of tools - tools that facilitate development, tools that assist in operation of application, tools that are embedded within application and used for

certain tasks, etc. We use some software tools which help during various phases of the project.

The tools used are mentioned below:

1. FreeTTS API for text-to-speech: We used this API because it's open source and also the implementation of Text-to-speech is simple. A voice package has to be downloaded and the JAR file has to be included as a library. Then FreeTTS objects can be used to call the voice.
2. Stanford dependency parser for conversion of phrase structure rules to dependencies: Stanford parser has to be downloaded and the JAR files associated with the parser has to be included in the application as a library. Then, documentation of Stanford parser has to be studied to see which classes and functions are important for conversion of phrase structure rules to dependencies.
3. Java Database Connect (JDBC) to connect MySQL with Java: To connect Java to MySQL database server, we need an intermediary which will help connect them. Java Database Connect is a software package which is developed by Oracle to map Java and MySQL. A JAR file has to be downloaded which is the driver for MySQL which has to be included in the application as a library. Then, API specific constructs can be used for forming and executing queries.
4. Netbeans IDE for coding in Java: Netbeans is a popular IDE used for developing Java applications. It's open source and supports Java and C++ along with other technologies like HTML and XML.
5. Netbeans IDE's swing palette for designing a UI: Netbeans provides an inbuilt palette for building GUI by using drag-and-drop.
6. Xampp server: Xampp contains a package of various servers like Apache, Tomcat, and also supports PHP, MySQL and Perl. We use Xampp for MySQL database.
7. Notepad++: This is an editor which supports programming in various languages including C, C++, Java, Python, HTML, C#, PHP, etc. We have used this editor for writing scripts of AIML.
8. StarUML: This is a tool for designing various UML models. It's proprietary and requires the user to purchase a licensed version after a trial. However, that's only when we want all the functionalities. We used it as a free trial for making our models.
9. Dia: This is also a tool for designing various diagrams. Unlike StarUML, this tool is open source and also provides variety of diagram models including UML diagrams.

5.4 Deployment diagram



Chapter 6: Integration and Testing

6.1 Testing approach

The testing approach involves the use of both the black box and white box testing.

In context to black box testing, we have given the users the access to the system. They have given different sets of input and the systems output was mapped accordingly to the expected results.

On the other side, the white box testing was done by the team members. We collected the data and inspected the code to see that output is given as expected or not.

6.2 Testing Plan

We will be testing following modules of the system:

1. Normalization module
2. POS Tagging module
3. Phrase Structure Parsing module
4. Phrase Structure to Dependency Conversion
5. Relation Extractor module
6. Relation Matching module
7. Relation to Sentence Matching module
8. Authentication module
9. Email sending module
10. Registration module
11. Voice output module
12. Viewing balance sheet
13. Applying for a new cheque book
14. AIML Parsing Module
15. Pattern Matching Module

Factors which will be considered during testing are *performance with respect to time* and *accuracy with respect to data available*. Achieving performance with respect to time is important as we need the modules to execute the tasks in least time. Accuracy with respect to data is important because we want the modules to give correct or expected results when inputs are given related to the data domain.

This will be followed by testing of the system after the integration of all the modules. The testing will be conducted by the team (Kiner Shah, Darshan Shah, Mohit Shetty) and each Pass / Fail result will be recorded. The Pass / Fail result will indicate what modules need to be improved to satisfy the tests.

6.3 Unit test cases

Test Name	**Test date**	**Test Input**	**Test Output**	**Expected Output**	**Test result**
Case Conversion	1st April 2017	“Account”	“account”	“account”	PASS
Spell Correction	1st April 2017	“Acccount”	“account”	“account”	PASS
Spell Correction	1st April 2017	“accountbalance”	“counterbalance”	“account balance”	FAIL
POS Tagging	1st April 2017	“a computer is a machine.”	a/DT computer/NN is/VBZ a/DT machine/NN ./.	a/DT computer/NN is/VBZ a/DT machine/NN ./.	PASS
Phrase Structure Parsing	3rd April 2017	“a computer can not be expected to give high performance.”	(S (NP (DT a) (NN computer)) (S' (VP (MD can) (VP (RB not) (VP' (VB give) (NP (JJ high) (NN performance))))) (. .)))	(S (NP (DT a) (NN computer)) (S' (VP (MD can) (VP (RB not) (VP' (VB give) (NP (JJ high) (NN performance))))) (. .)))	PASS

Phrase to Dependencies	6th April 2017	(S (NP (PRP You)) (VP (VBP do) (RB not) (VP (VB have) (NP (JJ much) (NN balance)))) (. .))	nsubj(have-4, You-1) aux(have-4, do-2) neg(have-4, not-3) root(ROOT-0, have-4) amod(balance-6, much-5) dobj(have-4, balance-6)	nsubj(have-4, You-1) aux(have-4, do-2) neg(have-4, not-3) root(ROOT-0, have-4) amod(balance-6, much-5) dobj(have-4, balance-6)	PASS
Relation extractor	8th April 2017	dep(provide-3, Does-1) nsubj(provide-3, bank-2) root(ROOT-0, provide-3) compound(cards-5, credit-4) dobj(provide-3, cards-5) case(people-7, to-6) nmod(provide-3, people-7) case(income-10, with-8) amod(income-10, less-9) nmod(people-	TYPE-BOOL,Characteristic,provide bank,Subject, null credit-cards,Object, null less,Characteristic,income provide,Action,null people,Prep_with,income provide,Prep_to,people	TYPE-BOOL,Characteristic,provide bank,Subject, null credit-cards,Object, null less,Characteristic,income provide,Action,null people,Prep_with,income provide,Prep_to,people	PASS

		7, income-10)			
Relation Matching	8th April 2017	TYPE-BOOL,Characteristic,provide bank,Subject,null credit-cards,Object,null less,Characteristic,income provide,Action,null people,Prep_with,income provide,Prep_to,people	provide,Action,null bank,Subject,null provide,Bool_neg,null provide,Prep_to,people less,Characteristic,income people,Prep_with,income credit-cards,Object,null Score = 57	provide,Action,null bank,Subject,null provide,Bool_neg,null provide,Prep_to,people less,Characteristic,income people,Prep_with,income credit-cards,Object,null Score = 57	PASS
Relation to sentence mapper	8th April 2017	provide,Action,null bank,Subject,null provide,Bool_neg,null provide,Prep_to,people less,Characteristic,income people,Prep_with,income	The bank does not provide credit cards to people with less income.	The bank does not provide credit cards to people with less income.	PASS

		credit-cards,Object,null			
Authentication: Username and password	15th April 2017	User inputs his username and password	Enables OTP button	OTP button should get enabled	PASS
Authentication: OTP	15th April 2017	User gets OTP via mail and enters it	Open new Home screen with User shown as Logged In	A new screen should open showing user as Logged In	PASS
Email sending	21st April 2017	Send OTP to user via Email	OTP message was sent to user's Email	OTP message should arrive at user's Email	PASS
Registration	16th April 2017	User inputs all details	A success message appears	A success message should appear or error message mentioning the error	PASS
Voice output	22nd April 2017	A text is input	Voice output corresponding to input text	Voice output corresponding to the text entered should be heard	PASS
Viewing	23rd	Appropriate	Transactions	Transactions	PASS

transactions	April 2017	options are selected as parameters	corresponding to those parameters	corresponding to those parameters should be shown	

6.4 Integrated system test cases

Test Name	**Test date**	**Test Input**	**Test Output**	**Expected Output**	**Test result**
Query of account type entered without login	24th April 2017	A query of account type	Error message asking user to log in	Error message should be shown asking user to log in	PASS
Query of general type entered	24th April 2017	A query of general type	Response corresponding to that query	Response corresponding to that query should be shown	PASS
Query of account type entered after login	24th April 2017	A query of account type	Response corresponding to that query	Response corresponding to that query should be shown	PASS

Chapter 7: Conclusion and Future Work

Thus, we have implemented the two approaches towards building the banking assistant - one using NLP and other using AIML. We thus conclude that NLP approach focussed much on analyzing the user's queries by performing series of operations like normalization, parsing, relation extraction and matching of relations. This gave good results on the sample data on which it was tested. AIML is a popular language for building chatbots and it gives results quickly by using pattern matching algorithm by means of various tags defined like pattern tag, template tag, etc. It doesn't try to analyze any query but simply matches patterns and gives results.

Our current architectures don't include much of the learning module which will keep on enhancing the system based on its experience with the users. If we include learning module and design it optimally, the overall system will interact with users more accurately, help them achieve their goals with more accuracy and thus will create a positive impression of system.

Some possible alternatives in the architecture of first approach can be inclusion of learning module in the class determination of the queries (general or account-related). For this, a simple Support Vector Machine can lead to good results after training it on a sample data of appropriate size. Other alternative can be to include learning in the relation matcher module wherein instead of matching based on scores, we can again use a Neural Network or a SVM classifier to classify the input relations to appropriate relations in the database.

References

[1] J. Bang, H. Noh, Y. Kim and G. G. Lee, "Example-based Chat-oriented Dialogue System with Personalized Long-term Memory," in *International Conference on Big Data and Smart Computing (BigComp'15)*, 2015.

[2] A. Trilla, "Natural Language Processing techniques in Text-To-Speech synthesis and Automatic Speech Recognition," [Online]. Available: http://atrilla.net/data/files/micnlp09.pdf. [Accessed 20 May 2017].

[3] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of 25th International Conference on Machine Learning*, 2008.

[4] R. McDonald , F. Pereira, K. Ribarov and J. Hajic, "Non-projective dependency parsing using spanning tree algorithms," in *Proceedings of the conference on Human Language Technology and Empirical methods in Natural Language Processing*, 2005.

[5] M. Zhu, Y. Zhang, W. Chen, M. Zhang and J. Zhu, "Fast and Accurate Shift-reduce Constituent parsing," *ACL*, pp. 434-443, 2013.

[6] D. L. Pennel and Y. Liu, "Normalization of text messages for text-to-speech," in *IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2010.

[7] S. Negi, S. Joshi, A. K. Chalamalla and V. L., "Automatically Extracting Dialog Models from Conversation Transcripts," in *Ninth IEEE International Conference on Data Mining*, 2009.

[8] L. Rabiner and B. Juang, "An Introduction to Hidden Markov Models," *IEEE ASSP Magazine,* vol. 3, no. 1, pp. 4-16, 1986.

[9] S. J. du Preez, M. Lall and S. Sinha, "An Intelligent web-based voice chat bot," in *EUROCON*, 2009.

Appendix

Appendix 1: Minimum system requirements

A. PC or laptop 32/64 bit window XP or higher with 2GB RAM.
B. Java Plug-in Software.
C. Internet Availability.

Appendix 2: User's Manual

To use the assistant for inquiries, follow the following steps:

1. Run BankingBot.java
2. A Home screen will appear. There type the query in the text box and click the "Submit" button. The response will appear in the text area below the "Submit" button.
3. If a message appears saying "Sorry, you have to log in". Then, click on the "Login" button from the right panel. A new window will open. There type your "Username" and "Password" and click on "Submit" button. The text box below the button will get enabled and you will get a message on the right of text box saying "Check you mail". Log in to your Email account. You must have received an email in your inbox. Copy the OTP code from the mail and paste it in the text box for OTP and click on Submit.
4. Now, after login, if you wish to perform certain operations like viewing transactions, or applying for a new cheque book, either you can select an option from the panel on the right, or you can type the query in the Query box and Submit the query.

Appendix 3: Technical Reference Manual

First approach:

Using Stanford Dependency Parser API:

1. Download the Stanford Dependency Parser.
2. Unzip the downloaded file. There should be a folder extracted.
3. Now, select four JAR files from the folder namely, stanford-parser.jar, stanford-parser-3.7.0-javadoc.jar, stanford-parser-3.7.0-models.jar, and stanford-parser-3.7.0-sources.jar, import them in the project libraries.

4. Open your browser and open the documentation for JavaNLP. There select the package named “edu.stanford.nlp.parser.lexparser” and there see the examples at the bottom of the page for reference on how to use different classes.

Using Java Mail API:

1. Download, the mail API.
2. Select JAR file named mail.jar and import in the project libraries.
3. Follow the documentation to use different classes in order to send Emails.

Using FreeTTS API:

1. Download FreeTTS API.
2. Follow the tutorial in this video lecture to learn how to use the API. https://youtu.be/swuYhvwHw9w.

Use JDBC Driver to connect to database:

1. Download JDBC driver for MySQL.
2. Import the JAR file into the project libraries.
3. Refer to documentation to know about available classes to connect Java and MySQL.

Second Approach:

1. Download Program O.
2. Extract Program O files to the server root or to the folders you want. (Assume we have extracted the file to folder ‘chatbot’, so that we access the program o by navigating through url http://example.com/chatbot/ when installation has completed.)
3. Create a database. If you’re using shared hosting you can create MySQL database in your host cpanel. Here we have used XAMPP Server.
4. Now go to http://example.com/chatbot/install, a installation page will appear and fill all the Botmaster info, Bot Configuration and Database Configuration.
5. You will be prompted to next page saying installation is successfull. Now goto http://example.com/chatbot/admin and enter the username and password that you have entered in Admin Area Username and Password. It will navigate to program o index page.

6. Create AIML files (Sublime Text 3 Editor, used) & upload the aiml files to Upload AIML section.
7. Now goto ‘test your bot’ section and see for yourself.

Papers published / Project competitions

We have published a paper in a journal - International Journal of Computer Applications:

K. B. Shah, D. P. Shah, M. S. Shetty and R. Pamnani, "Approaches towards Building a Banking Assistant," *International Journal of Computer Applications*, vol. 166, no. 11, pp. 1-6, 18 May 2017.

We have also participated in a project competition, called, Prakalpa, which was a state-level project competition organized by our college. The published paper and the certificates of the project competition are attached in the next pages.

Plagiarism report