ASHESI UNIVERSITY COLLEGE

AUTOMATING CHALKBOARD SUPPORT PROCESSES USING A CHATBOT

Applied Project
B.Sc. Management Information Systems

Joseph-Peter Yoofi Brown-Pobee
April 2019

ASHESI UNIVERSITY COLLEGE

Automating Chalkboard Support Processes using a Chatbot

Applied Project

Applied Project submitted to the Department of Computer Science, Ashesi University College in partial fulfillment of the requirements for the award of Bachelor of Science degree in Management Information Systems

Joseph-Peter Yoofi Brown-Pobee
April 2019

DECLARATION

I hereby declare that this Applied Project is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:

……

Candidate's Name:

……

Date:

……

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dennis Owusu for the splendid feedback and guidance he has given me throughout the period. I am also grateful to Chalkboard Education for giving me key information necessary to the completion of this project

ABSTRACT

This project seeks to apply natural language processing techniques to automate the support processes of Chalkboard Education, a startup in Ghana. The company has many users but currently has only two operations personnel responsible for responding to customer queries. Their support processes can be made easier with the use of a chatbot. The project references work from early chatbots like ELIZA and Cleverbot as well as more recent ones like MOOCBuddy and SuperAgent. The chatbot is built on RASA Natural Language Interpreter and uses third party APIs like Twilio and Database connections to mirror human support processes. Upon implementation, the chatbot is suitably able to perform the repetitive human tasks the operations personnel were carrying out, successfully and in shorter time. In the event that the chatbot cannot handle a query, the problem is forwarded to the aforementioned operations personnel. Upon evaluation, it was discovered that the chatbot has could improve its accuracy and effectiveness with techniques such as adding more training data and using different language models for embeddings. In the future, the chatbot can be implemented via a USSD application to enable Chalkboard capitalize on the prevalence of USSD application in Africa.

TABLE OF CONTENTS

Chapter 1: Introduction

Many businesses in Ghana make use of customer service call centers to respond to the queries of customers and resolve any issues they may have. A simple search on-line would produce a list of jobs available at said call centers. In principle, these centers are very instrumental to the business operations as they serve as a contact point to customers who are revenue drivers for the business. However, practicality dictates otherwise. In some cases, the numbers provided for customer service are no longer in service (some have been changed but not updated). In other cases, the call experience is marred by the emotional state of calling customers or the receiving agents at a given point in time. Even beyond these issues exist a more significant problem of cost. For companies, as they scale up and acquire more customers, there is a need to hire more customer service personnel to cater to the needs of customers.

An already existing solution to this problem is the use of automated conversational systems to respond to the queries of customers. Intelligent systems exist that can quickly serve numerous customers at any given time before deciding if human intervention is needed. Systems differ to varying extents based on the contents of their domain. A domain, in this case, is merely the scope of questions, statements, and classes that all communication can be categorized under. Hence, even though many automated systems function similarly, they need to be built for a specific domain to be effective.

Chalkboard Education is a startup that hosts an education platform. The company allows schools to make all their content digital and enrolls their students onto a platform to access it. The school gives their course material to Chalkboard Education who digitize it, upload it to the platform and create accounts for all the students who can then access the content from their mobile phone via a 6-character login token. Problems with the platform are thus handled solely by Chalkboard and not the school. This results in many individual

students always calling and direct messaging Chalkboard's customer service support line for help. The company currently has only two people available for this job. This project's main contribution is to automate some of Chalkboard's support processes using a task-oriented dialog agent to resolve common recurring user problems. By the end of this project, the goal is to build a full-functioning chatbot to handle all of Chalkboard's support issues.

1.1 Background & Motivation

The use of conversational agents and dialog systems can be traced as far back as 1966 with the development of ELIZA, an early natural language processing computer program to present day Siri and Alexa of Apple and Amazon respectively. Natural Language Processing and its related fields have provided a means for us, humans, to communicate with machines and machine systems in ways similar to how we do with each other. There are many systems with different implementations and nuances, but the majority generally fall in two classes: Task-oriented dialog agents and Chatbots.

Task-oriented dialog agents are predominantly concerned with specific tasks and designed to have as very little interactions as possible to accomplish the task [1]. Examples include digital assistants like the Siri above and Alexa. This class of dialog systems is mostly employed by companies on their websites and products to enable customers and users to address problems and answer questions. Key benefits of these systems, especially to businesses and their customers, include faster and more convenient query handling as well as lower costs relative to hiring of human agents. They are not designed to have prolonged conversations. However, chatbots, the second class of dialog systems, are designed for that.

Chatbots are set up to mimic the natural conversational characteristics of human beings. They are designed to have more casual and less directed conversations than task oriented dialog agents. Cleverbot is a chatbot capable of carrying on prolonged

conversations with humans and exists mainly for entertainment purposes [5]. These are more suited for social uses such as for psychological analysis and entertainment purposes. A form of evaluation for many chatbots is a test called the Turing Test developed by Alan Turing in 1950. A Turing Test is a method of inquiry for determining whether or not a computer/system is capable of thinking like a human being[2]. A human should be able to find the two systems indistinguishable to pass the Turing Test. A chatbot that can pass this test can be said to be a good one.

The existence of the above systems in the field of natural language processing have made it possible for the application of knowledge to different scenarios and domains. Chalkboard Education's particular situation presents an opportunity to apply this knowledge in a local context. The startup has been running for close to three years and currently has over 4000 students enrolled on its platform. Customer queries are handled mainly through WhatsApp and voice phone calls and require employees to be present to respond at all times. The creation of a dialog system using the information and queries specific to Chalkboard appears a potential solution to solve the problem by removing the need for the physical presence of human and saving costs for the startup. As the startup scales, the system does not have to scale proportionally; hence the cost of customer service can be minimal for a given number of users.

1.2 Related Work

1.2.1 ELIZA

ELIZA is an early natural language processing computer program created from 1964 to 1966 [3]. It was created by Joseph Weizenbaum at the MIT Artificial Intelligence Laboratory to demonstrate communication between humans and machines using natural

language. It works by breaking down a sentence, ranking its keywords and transforming the user's sentence into a proper response using rules from its pre-programmed learning script. The system was modeled after a Rogerian psychotherapist in that it allowed the program do not need to have a knowledge base of the topic the human was discussing [Weizenbaum, 1966]. By merely transforming the sentences based on keywords in the user's query, the program can appear to be engaging in conversation regardless of the topic. ELIZA would turn user input into a question which kept the users engaged. There were several reports of human test subjects developing an emotional connection to the system; a testament of how well it was able to sustain a regular conversation. This system formed the basis for the creation of other conversational agents and improvements in the field of natural language processing.

1.2.2 Cleverbot

Cleverbot is a web-based chatbot that uses artificial intelligence to have unsupervised conversations with humans and is created by Rollo Carpenter. It holds the distinction of having passed the Turing Test. As mentioned earlier, the idea of the test is for a machine to pretend to be a human and will only pass if this pretense is found to be convincing [4]. Cleverbot has performed quite well in Turing Test competitions, giving it some credibility [4]. Unlike ELIZA, Cleverbot learns how to have a conversation as it interacts with more humans. Its responses at any given time are as a result of an analysis of previous conversations it has had with other humans. This contributes to its ability to perform well on the Turing Test. Despite the difference in how ELIZA and Cleverbot are implemented, they are both capable of holding a conversation with a human for a considerable length of time.

1.2.3 Super Agent: A Customer Service Chatbot for E-commerce websites

This paper highlights the need for the use of automated systems like chatbots in a business environment. It labels customer service as one of the most resource intensive departments within a company, consuming a lot of time and money. Customer queries are repetitive, and customer service agents cannot be present 24/7. Both of these can be resolved through the use of chatbots as the paper points out. Chatbots are economical, indefatigable and would enable support staff to spend more time on other things. The bot created in the paper, SuperAgent, leverages 'large-scale and publicly available e-commerce data' to respond to customer queries. A key contribution of the paper to this proposed project work was how to involve a 'chit-chat engine' to satisfy usability in terms of customer experience and achieve the non-functional requirement of naturalness of conversation. SuperAgent uses a chit-chat engine to reply to queries that cannot be answered by all other engines and to respond to small talk off-topic user inputs. The provision of this engine helps push the conversation with the chatbot to look closer to that of a customer service agent.

1.2.4 MOOCBuddy

MOOCBuddy is a chatbot that serves as a recommender system to help users find the best courses on Massive Open Online Courses like Coursera and edX. It is created based on the user's social media profile and is managed through Facebook Messenger. The paper by Holotescu [8] tells of how the creation of Facebook's Messenger service, which leverages billions of individual users and businesses on Facebook, has led to the increase in the number of chatbots available. These chatbots are built and run through the Messenger API. Dialogues are modeled as structured messages with URLs connected to enable users to make choices. It also offers users the ability to search using topics, languages, and dates among others. This paper ultimately serves as an example of the use of chatbot for an online

educational platform built with Facebook Messenger. This is similar to what is being done for Chalkboard Education throughout this applied paper and hence serves as a basic proof of concept.

1.2.5 E-business Chatbot Using AIML and LSA

This work proposes building a chatbot using a combination of Artificial Intelligence Markup Language(AIML) and Latent Semantic Analysis (LSA) as a solution for improving customer service. Similar to other works on the topic, the paper outlines some of the inadequacies of humans in the customer service department and the great benefits of employing an automated system. AIML is a dialect used for creating natural language system agents. They use AIML to handle general queries like ‘Whats up' and ‘hello.' It cannot be used for more specific queries because it requires the developer to anticipate all the specific ways the user might express an intent. LSA is used here to find the similarity between words in vector representation form. The combination of these two can help the chatbot understand user input and generate a suitable response. Inputs are first handled by the AIML, and if the input exists in the templates, the appropriate prepared answer is given. If not, it is passed to the LSA to produce a semantic-based answer. The LSA is trained on FAQs from the given business. The main benefit of the paper to the project is to help understand and view other implementations of chatbots, validate the problem-solution fit once again and to understand the flow of processes a chatbot might go through to produce a response.

Chapter 2: Requirement Specification

The chapter seeks to give an analysis of the functionality the task dialog system would offer as well as the scope of its capability. The application would make use of frequently entered user inputs to learn and generate appropriate responses to user queries. The learning and generation of responses will be done without human supervision. Requirements would be obtained from Chalkboard Education as well as literature such as Speech and Language Processing by Daniel Jurafsky [1] to identify the necessary components for building a task dialog system. At various parts of this document, the system may be referred to as a bot.

2.1 User Identification and Use Case

The application would be used primarily by students of schools that use Chalkboard Education's platform to manage their content. The service would run on Facebook Messenger; hence users would need Internet access and a subscription to a mobile carrier to access the dialog system. The primary use cases for which the chatbot is being built are outlined below to understand better how the application will be used:

1. A student of a newly added school (freshly added to Chalkboard's platform) attempts to access the platform but does not know how to log in/does not have their log in details.
2. A student of a newly added school can log in but does not know how to navigate the site and access resources.
3. A student of an already existing school on the platform is unable to find specific course material on the platform.

2.2 Procedure for Requirements Gathering

The requirements were obtained through the following means:

Interview: Interview with an Operations Associate at Chalkboard Education. Through this interview, we can identify the key problems that the dialog system needs to resolve and the extra Chalkboard specific information and resources the bot would need to accomplish the task.

Literature Review on Chatbots and Dialog Systems: Books such as Speech and Language Processing by Dan Jurafsky help identify some universal concerns and usability issues to address when building conversational systems in general. The concerns are not specific to Chalkboard but are more general to any conversational system being built.

2.3 Requirement analysis

The main function of the task dialog system is to provide appropriate generated answers to user queries through Facebook Messenger without human supervision. A key input is a set of previously asked queries to enable the system to learn queries and match them to appropriate responses. All use cases would involve the user querying the system.

2.3.1 Functional Requirements

- Users should be able to type in any problems they have or queries they need to be answered.
- The system should be able to read the queries as input and generate a response that best answers the query.
- The user should be able to read the generated response. The response should be in a form readily understandable by the user.
- The system should be able to process queries that have the same idea but appear in different forms and contexts.

- The system should be able to learn from real-time queries to be more capable of handling queries it has not been trained on
- The bot should be able to resolve login issues
- The bot should be able to resolve enrolment issues
- The bot should be able to resolve navigation issues
- The bot should be able to recognize registered users of Chalkboard
- The bot should be able to resolve user queries in less time than its human counterpart

2.3.2 Non Functional Requirements

- The bot should be able to respond to user queries in under 20 seconds.
- Talking to the bot should feel like talking to a human being.
- The bot should not give out sensitive information during the conversation.
- The system should be able to respond to a few non-problem oriented queries to maintain some level of interactivity with the user.
- The bot should be able to identify when it cannot solve a problem and refer to personnel.
- The bot should be able to work through simple spelling mistakes.

All requirements outlined above will be addressed in this project.

Chapter 3: Architecture and Design

3.1 High-Level Architecture of Project

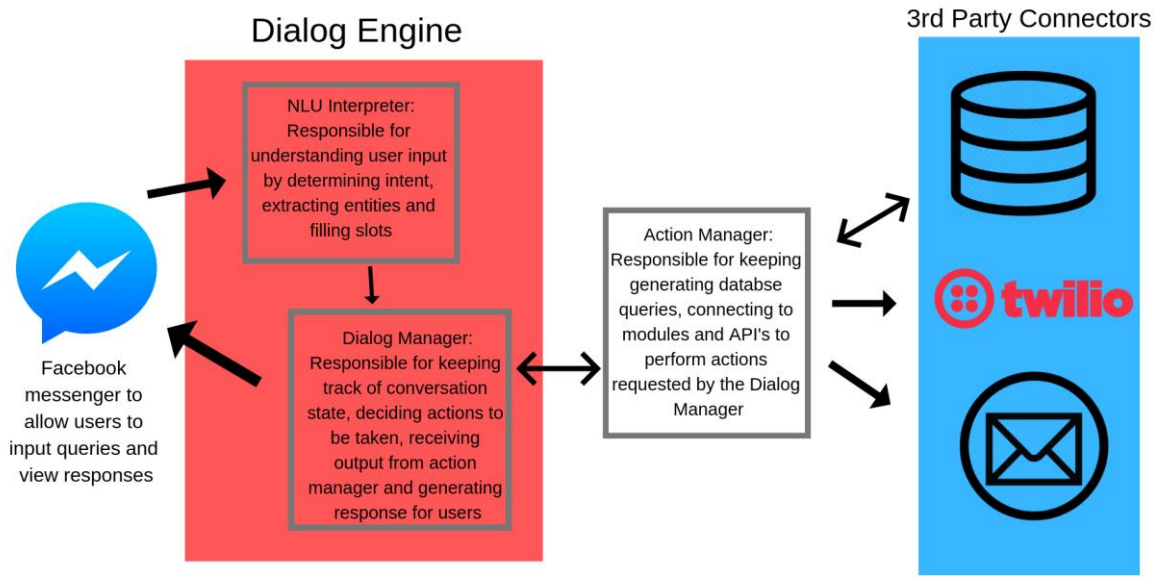


Fig 3.1

3.2 Key Modules in the Architecture/Design

3.2.1 Facebook Messenger Application

Facebook's Messenger Application will be used to receive inputs and send generated responses to users. The pervasiveness of Facebook and the large number of users who access it make this a suitable medium.

3.2.2 Dialog Engine

Below is a more detailed illustration of the architecture of the dialog engine component:

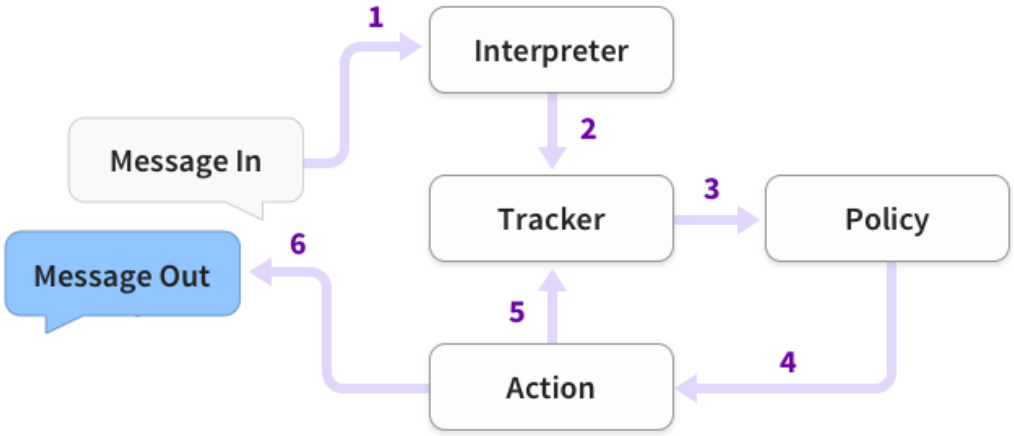


Fig 3.2
Source: https://rasa.com/docs/core/architecture/

3.2.3 NLU Interpreter

This component is responsible for understanding user input. For a given input, it breaks it down by words to identify the intent of the user. It does this using pre-trained sentences for given intents and compares similarity. It also extracts entities from the input. Entities are key objects that are useful when executing actions and generating responses. The interpreter is also responsible for slot filling. Slots represent information required for certain actions. For instance, if a user wants to enroll for a course, a slot to be filled is the user's name and the course name.

Training with the NLU Interpreter

Intent Classification

RASA can use different pipelines to process user messages. A pipeline defines different components which process a user message sequentially and ultimately lead to the classification of user messages into intents and the extraction of entities [10]. The two most important pipelines RASA uses are tensorflow_embedding and spacy_sklearn[11]. spacy_sklearn is more suitable for this project because it uses pre-trained word vectors

which is very useful when training data is limited. Using this pipeline, RASA takes each piece of training data and its marked intent to create a training data object. This object has the text of the document, its intent and the entities marked as keys in the object. RASA uses a Sklearn [15] Intent Classifier. The classifier uses spaCy, an open source library for natural language processing, to convert each training object into a list of tokens(words), creating a bag of words. This is referred to as tokenization. RASA then moves to feature this bag of words by converting the tokens into word vectors. These are known as Word embeddings which can capture semantic and syntactic aspects of words [10]. Machine Learning Algorithms understand numerical data which is why the words need to be converted to word vectors. The features are then labeled with the intents they describe in a numerical format. Hence instead of labels like [login, list_courses], numbers will be assigned to represent the labels.

The Sklearn Intent Classifier uses GridSearch [15] with intent names as labels and features as those generated by the feature to generate a Machine Learning Model. The Machine Learning Model is a mathematical representation of a real word process; in this case, determining whether a given text belongs to a particular intent. GridSearch is a hyperparameter tuning algorithm that helps identify the optimal hyperparameter to use in a machine learning model. It works by building and evaluating a machine learning model for each combination of parameters specified. Hyperparameters are values that help a model produce the most accurate predictions but cannot be estimated from the data. Selecting hyperparameters can be likened to tuning a radio for the right frequency. Once this is done, it stores the model in a persisted file. The final output is an Interpreter object which can classify an intent based on the model. When a new input is provided, the Interpreter returns a numerical value based on the model trained which needs to be decoded to arrive at the

actual intent tag. A set of confidence values are created for each prediction, and the highest confidence value is selected as the intent.

Entity Extraction

RASA uses Named Entity Recognition using Conditional Random Fields (NER_CRF) as the algorithm for entity extraction [11]. Named Entity Recognition is used to identify and classify words in a document into defined categories/labels. [12]. Conditional Random Fields are used to predict the most likely sequence of labels based on a sequence of inputs [13]. Here, again, as part of preprocessing, the text is tokenized but in this case stop, words are not removed as is the case in other NLP techniques. This is because, with NER_CRF, every feature depends on features preceding and succeeding it. The training data is converted with entities to a list of tuples with a start_index, end_index, and entity. So for instance, the marked training data "I am Dennis[first_name]" would have a tuple of [(5,10, 'first_name')] to denote where the entity starts and ends in that sample. The range (5, 10) is regarded as the offset of the entity. This helps tag tokens from the preprocessing step. BILOU (Begin, Inside, Last, Other, Unigram) tagging is a way of encoding information in a set of labels by recognizing the Beginning, Inside and the Last token of data relating to entities and differentiate them from Other tokens and unigram tokens. It is used for each token whereby if the token falls within the offset of the entity tag, an entity tag is attached to it; if the token does not, it is given a value of 0. We end up with each token and its assigned entity in each training example. After this is completed each token is again analyzed to determine further characteristics about the word for training. Characteristics like the previous word, whether the previous words is a title, whether the current word is a digit and whether it is at the end of a sentence are taken. This information together with the information from BILOU tagging is used to train and fit a CRF model for prediction. During

prediction, the text entered by the user is broken into tokens and processed to arrive at the breakdown of its characteristics. This breakdown is then fed to the CRF model for classification. We obtain a list with the probability that a token has a BILOU tag. Tags with 0 are ignored. A new entry like ‘I am Jesse’ would result in a result in a result such as [{‘start’: 5, ‘end’:9, ‘value’:’Jesse,’ ‘entity’: ‘first_name’}].

Dialog Manager

This component is responsible for keeping track of the conversation state at all times. It makes use of a Dialogue State Tracker and Policy objects as seen in Fig 3.2. The Tracker stores and maintains the state of the dialogue with a single user [11]. Events describe everything that occurs in a conversation and are stored in the tracker. The Tracker stores Events such as:

- The user saying something to the bot.
- The bot saying something to the user
- The user specifying the value for a slot
- The bot restarting a conversation
- The conversation being paused and resumed
- An Action being executed or rejected
- The last action executing
- The number of state turns and intents made

The Tracker also stores current slot and entity values. It passes its state to the Policy object which decides what action to take at every step in the dialogue[11]. The Policy predicts the next action the bot should take after being passed the tracker. It produces a list of probabilities/confidence scores for the next actions based on the state of the Tracker and selects the one with the highest probability. The Policy can decide by training on the Trackers it receives. The Tracker provides a bag of active features which consist of Events

that have been recorded. Policy converts these features into vector representation with an array containing the target class labels encoded as one-hot vectors[11]. It does this using multiple Featurizers. For instance, the BinarySingleStateFeaturizer creates a binary one-hot encoding with the vector indicating the presence of an intent, entity, previous action or slot [11]. After featurizing, te Policy trains on the Tracker features and then predicts the action probabilities. Keras and Memoization Policies are some policies RASA uses. The former uses a neural network implemented in Keras to select the next action and the latter simply memorizes the conversations in the training data and predicts the next action based on this[11]. If the Policy used returns low confidence in multiple stages, a two-stage Fallback Policy is used which asks the user to first affirm the intent. If is affirmed, the conversation continues with the affirmed intent and if not the user is asked to rephrase their message. If the rephrased message produces high confidence, the conversation continues; if not, the user is asked to affirm again. If the user affirms the second time the conversation continues, if not a fallback action is executed by the Action Manager.

The action to be taken next is passed to the action manager. Upon executing the action, the action manager sends the output of the action to the dialog manager who then generates a response to be sent back to the user.

3.2.3 Action Manager

This component is responsible for generating and executing the necessary code for successfully executing actions. It works with a database by generating and executing necessary SQL queries to cull data. It also formats the data received from the database in a form suitable for the dialog manager. The Action Manager also interacts with the Twilio API to send text messages to a number provided by the Dialog Manager. Login URLs will be sent through this means. It also uses the Python SMTP Library to send an email to personnel at Chalkboard in the event the chatbot receives queries it is unable to handle. The

Manager stores all the tokens that are necessary to interact with these APIs. The Manager is also responsible for writing logs of conversations with users for storage.

3.2.4 Third Party Connectors

The Action Manager will interact with a MySQL database, the Twilio API and SMTP Library. This will enable it to extract database information, send SMS's and emails respectively.

3.3 Design Component to Requirements Mapping

Requirement	**Architecture component responsible for addressing requirement**
Users should be able to type in any problems they have or queries they need to be answered.	Facebook Messenger Front End Integration
The system should be able to read the queries as input and generate a response that best answers the query.	Dialog Engine
The user should be able to read the generated response. The response should be in a form easily understandable by the user.	Dialog Engine, Facebook Messenger Front End Integration
The system should be able to process queries that have the same idea but appear in different forms and contexts.	Dialog Engine (NLU Interpreter)
The system should be able to learn from real-time queries to be more capable of handling queries it has not been trained on	Action Manager
The bot should be able to respond to user queries in under 20 seconds.	Dialog Engine, Action Manager
Talking to the bot should feel like talking to a human being.	Dialog Engine
The bot should not give out sensitive information during the conversation.	Action Manager, Twilio API
The system should be able to respond to a few non-problem oriented queries to	Dialog Engine

maintain some level of interactivity with the user.	
The bot should be able to identify when it cannot solve a problem and refer to personnel.	Action Manager, Gmail API,
The bot should be able to work through simple spelling mistakes.	Dialog Engine (NLU Interpreter)
The bot should be able to resolve login issues	Dialog Engine, Action Manager, Database, Twilio API
The bot should be able to resolve enrolment issues	Dialog Engine, Action Manager, Database
The bot should be able to resolve navigation issues	Dialog Engine, Action Manager
The bot should be able to recognize registered users of Chalkboard	Dialog Engine, Action Manager, Database
The bot should be able to resolve user queries in less time than its human counterpart	Dialog Engine, Action Manager

Chapter 4: Implementation

Overview

This section intends to describe all the steps taken to build the dialog system and an accompanying application to mirror the Chalkboard website. The application will have an entry for students and administration, and the bot will make changes that are possible on the administration platform to be reflected on the student platform. For the application and the bot, the components used to build it and the functionality they provide will be explained to give a full picture of the implementation

4.1 RASA Core and NLU

RASA is an open source platform with tools for building virtual assistants and conversational systems. It features a Natural Language Understanding (NLU) and Core components. The former is used to help the bot understand what the user is saying while the latter is used to manage the flow of the conversation and the performing of corresponding actions.

4.1.1 RASA NLU

The Natural Language Understanding tool is essential for intent classification and entity extraction. For a given sentence, the NLU helps identify what the user wants to do and the resources available to do it.

Defining Intents

When building the bot for this system, the first step was to define the intents for the system. Intents represent what the students seek to do with the system. For instance, if a user types ‘I am unable to log in,' the intent here is to report difficulty with logging in. I defined intents for our system listed below:

- greet: When a user intends to greet the bot
- goodbye: When a user intends to leave the bot
- thanks: When a user intends to express gratitude
- deny: When a user seeks to answer negatively to a question or statement
- give_course: When a user seeks to give the name of a course
- give_name: When a user intends to give their name
- login: When a user intends to report an issue about logging in
- navigation: When a user intends to report an issue about navigation and generally using the application
- enrollment: When a user intends to report an issue about enrolling
- affirm: When a user seeks to answer position to a question or statement
- list_course: When a user seeks to see a list of all the courses being offered
- introduction: When a user seeks to introduce the
- age
- beautiful
- birthday
- boss
- help
- good
- hobby
- occupation
- origin
- how_are_you
- my_birthday

For each intent, the bot needs several sample sentences that show how a user might express the various intents. The bot needs this to be able to train to identify an intent given a new sentence. Hence for each intent, sample sentences where given. Below are some of the sentences given for the ‘login' intent

intent:login
- I cannot log in
- Cannot log in
- I have login issues
- I don’t know where to find my URL
- Can you send me my login URL
- Cannot find URL
- Cannot log in
- Where is my link
- There are supposed to be six characters. How do I get my six characters
- I cant see my log in link
- I cannot see my six characters
- My URL is not working
- I did not get a text message from you
- I did not receive an SMS

This is done for all intents listed above.

Marking Entities

For some intents, there is a need to mark entities in the sample sentences to enable RASA identify them when new input is provided. An entity is any item of interest that can be used to perform some further action. For the ‘give_name’ intent, the sample sentences have been marked as shown below:

```
# intent:give_name
- My name is [Yoofi](name) [Brown-Pobee](last_name)
- I am [Ebenezer](name) [Lamptey](last_name)
- I'm [Elvis](name) [Boateng](last_name)
- People call me [Sarah](name) [Agyapong](last_name)
- It's [Judith](name) [Asaaba](last_name)
- Usually people call me [Julian](name) [Adusei](last_name)
- My name is [Adam](name) [Serwaa](last_name)
- You can call me [Abigail](name) [Adamtey](last_name)
- Please call me [Mary](name) [Brakoh](last_name)
- Name name is [Blankson](name) [Frimpong](last_name)
- I am [Aku](name) [Nsowine](last_name)
- I'm [Adel](name) [Amponsah](last_name)
- Call me [Faustina](name) [Adjei](last_name)
```

The entities used here are ‘name’ to represent the user’s first name and ‘last_name’ to represent their last name. The bot for the Chalkboard system requires three entities:

- name
- last_name
- course_name

Hence in the ‘give_course' intent I use the sample sentences marked as shown below:

```
#intent:give_course
- The course is [English with Elements of Literature](course_name)
- The course is [LITERATURE IN FRENCH II](course_name)
- The course is [LITERATURE IN FRENCH](course_name)
- The course is [ENVIRONMENTAL AND SOCIAL STUDIES I](course_name)
- The course is [ENVIRONMENTAL AND SOCIAL STUDIES](course_name)
- The course is [MATHEMATICS II (GEOMETRY AND TRIGONOMETRY )](course_name)
```

- The course is [MATHEMATICS](course_name)
- The course is [GEOMETRY AND TRIGONOMETRY](course_name)
- The course is [CHILD AND ADOLESCENT DEVELOPMENT AND LEARNING](course_name)
- [JEN 123](course_name)
- [JMC 122 MATHEMATICS II (GEOMETRY AND TRIGONOMETRY)](course_name)
- [METHODS OF TEACHING RELIGIOUS AND MORAL EDUCATION](course_name)
- [CHILD AND ADOLESCENT DEVELOPMENT AND LEARNING](course_name)
- The course is [Literature in French](course_name)

Based on the entities I create slots to let RASA know the kind of data to expect and how to store them. The entity names and slot names must match for RASA to be able to extract the information to fill the slots. These slots are used to store entities to perform other actions. The slots are defined as:

name:
    type: text

last_name:
    type: text

course_name:
    type: text

The above means there three slots to fill and each would have a data type of text.

Once the intents, entities, and slots have been defined, the next step is to train the NLU on the intents and entities. The NLU proceeds to learn the various ways a user can express intents and entities to process entirely new input when asked.

4.1.2 RASA Core

Creating Stories

The Core is responsible for managing the conversation and performing corresponding actions. With the core, the first step is to create stories which represent possible conversations the bot might have with users. An example story that the Chalkboard Bot would find useful is

```
## story_login_01
* login
  - utter_ask_for_name
* give_name{"name":"Ayorkor", "last_name":"Brown-Pobee"}
  - action_send_token
  - utter_useful
* affirm
  - utter_thanks
```

In the above, the name of the story is ‘story_login_01,' and the first thing that would trigger the story is the user expressing the intent ‘login.' Once this is expressed, the bot will execute the action (more on actions in the subsequent sections) ‘utter_ask_for_name’ which would ask the user for their name. The bot expects the user to give it a name as shown by the ‘give_name’ intent. The ‘name’ and ‘last_name’ in the curly braces represent entities for the bot to look out for. The names by them are placeholders. The bot knows to expect some names. After giving the name, the chatbot runs the action ‘action_send_token.’ This action checks the names given against the database available and returns the token for the given student. The user is then asked if they found the solution they were given was useful. In this story, the user answers in a positive manner with the intent ‘affirm’ and the chatbot thanks them using the action ‘utter_thanks

I define different variations of stories for a single intent to capture the different ways the conversation with the user might go similar to below:

```
## story_login_01
* login
  - action_write_log
  - utter_ask_for_name
* give_name{"name":"Ayorkor", "last_name":"Brown-Pobee"}
  - action_write_log
  - action_send_token
  - utter_useful
* affirm
  - action_write_log
  - utter_thanks
## story_login_02
  * login
    - action_write_log
    - utter_ask_for_name
```

```
* give_name{"name":"Ayorkor", "last_name":"Brown-Pobee"}
  - action_write_log
  - action_send_token
  - utter_useful
* deny
  - action_write_log
  - utter_refer_to_person
  - action_send_email
```

Login story 01 handles what would happen if the action carried out by the chatbot is successful while login story 02 handles the negative situation. This makes RASA more flexible and more natural.

Many stories are defined to try to capture the multiple ways users may interact with the chatbot. This is suitable for Chalkboard because the actions that can be taken are finite; hence there are only a few known things students can want to do. There is an action to send an automated email to Operations in the event of a persistent problem thereby ensuring only the most important need human intervention.

Actions

There are different actions that carry out various results. Some actions are simply response actions that display text to the user. These are utter actions as shown below:

```
utter_name:
- text: "Hey there! Tell me your name."

utter_greet:
- text: "Nice to you meet you {name}. How can I help?"

utter_goodbye:
- text: "Talk to you later!"

utter_thanks:
- text: "My pleasure."

utter_ask_for_name:
- text: What's your full name
```

utter_full_name:
- text: "Your first name is {name} and your last name is {last_name}"

utter_useful:
- text: "Was that useful?"

utter_try_again:
- text: "Okay, {name} lets try this again"

utter_refer_to_person:
- text: "Looks like your issue is a bit more nuanced and I will forward it to Operations to get in touch with you shortly"

utter_restart:
- text: "Anything else"

utter_ask_for_course:
- text: "What course is this"

These are the most commonly used as they help the robot maintain a conversation with and direct the user. More complex Actions are then defined to make the chatbot more useful to the user. These actions make use of Python classes. The class has two methods: a name method and a run method. The name method returns the name of the action as specified in the domain file. The name here must match the name in the domain file for the Dialog Engine and Action Manager to communicate successfully. When the name method is called and returns successfully, the run method is called to execute the action. For instance, the action 'action_send_token' takes the first name and last name of the user from the slots activates the Twilio API and sends the user their login link via SMS. Another action is the 'action_send_email' action that sends an email to Chalkboard Personnel when the chatbot is unable to handle a user query

```
class ActionSendEmail(Action):
    def name(self):
        return "action_send_email"

    def run(self, dispatcher, tracker, domain):
        output = open('Output.txt')
        smtpserver='smtp.gmail.com:587'
        from_addr = 'insightnetwork.15@gmail.com'
        to_addr_list = ['insightnetwork.15@gmail.com']
```

```
        cc_addr_list = ['']
        subject = "Customer query I cannot handle"
        message = output.read()
        login = 'insightnetwork.15@gmail.com'
        password = 'fqpvbcjkwunvzqdk'
        header  = 'From: %s\n' % from_addr
        header += 'To: %s\n' % ','.join(to_addr_list)
        header += 'Cc: %s\n' % ','.join(cc_addr_list)
        header += 'Subject: %s\n\n' % subject
        message = header + message

        server = smtplib.SMTP(smtpserver)
        server.starttls()
        server.login(login,password)
        problems    =    server.sendmail(from_addr,    to_addr_list,
message)
        server.quit()
        date = datetime.datetime.today().strftime('%Y-%m-%d')
        f_name = tracker.get_slot('name')
        l_name = tracker.get_slot('last_name')
        os.rename("Output.txt",                    "Errors/"+f_name
+"_"+l_name+"_"+date+"_"+".txt")
        open('Output.txt', 'w').close()
        return problems
```

After each query the chatbot runs the 'action_write_log' action to record the utterances of the user to an output file and stores it. This file is used to store user sentences for training the NLU to be more accurate. It is also used when the chatbot cannot handle a query and needs to be referred to personnel as shown above. In this way, the chatbot can learn from new queries and to let personnel know of difficult situations.

Below is a summary of all some of the actions defined and what they accomplish:

utter_name: Asks the user for their name
utter_thanks: Says thank you to the user
used to test that the chatbot was capturing the first name and last name from the full name given.
action_send_token: Interacts with the Twilio API to send the user their login link through SMS
action_enrol_student: Enrols the student in a course so they can see course material on their dashboard
action_list_all_courses: Lists all available courses for a student
action_send_manual: Sends the user a pdf form of the walkthrough manual directly as an attachment
action_write_log: Records user utterances and query into a text file and saves it
action_send_email: Sends the content of conversation log as an email to Chalkboard personnel

Actions are declared as python classes with the main action to be accomplished declared as a run method as seen above.

Every single nontext action that has to be performed is declared in a Python file called actions.py.

Domain

After the actions, stories, intents, entities, and slots are defined, a domain file is created that contains all necessary defined components the chatbot needs to be aware of. If a component is defined in the domain file but not elsewhere, RASA would report an error. In the same way, if a component is defined elsewhere but not in the domain file, it would not be functional. All actions, stories, intents, entities, and slots go into the domain file. Below are some of the contents of the domain file:

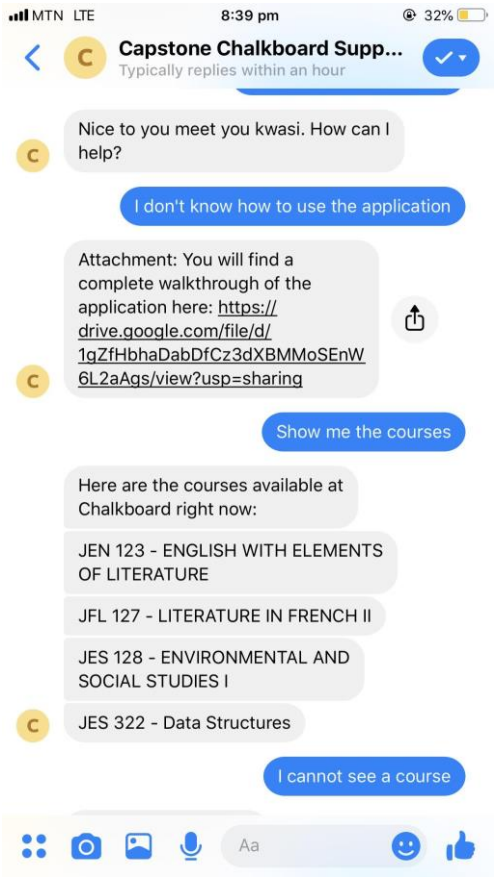
```
intents:
- greet
- goodbye
- thanks
- deny
- give_course
- give_name
- login
- navigation
- enrollment
- affirm
- list_course


entities:
- name
- last_name
- course_name


slots:
  name:
    type: text
```

```
  last_name:
    type: text

  course_name:
    type: text


actions:
- utter_try_again
- utter_refer_to_person
- utter_ask_for_course
- action_enrol_student
- action_list_all_courses
- utter_send_manual
- action_send_manual
- action_write_log
- action_send_email


templates:
  utter_name:
  - text: "Hey there! Tell me your name."

  utter_greet:
  - text: "Nice to you meet you {name}. How can I help?"

  utter_full_name:
  - text: "Your first name is {name} and your last name is {last_name}"

  utter_useful:
  - text: "Was that useful?"

  utter_try_again:
  - text: "Okay, {name} lets try this again"
```

Facebook Messenger routing

A page is created on Facebook, and the Messenger Platform added, to enable it to use Messenger to send and receive messages. Credentials were generated to grant the chatbot

application permission send and receive messages through Messenger. A webhook with a callback URL is set up to receive facebook messenger messages. The output is shown below.



API and Third Party Application Implementation

Twilio API

An account is set up with Twilio which is a cloud communications platform that allows developers to make and receive phone calls and text messages using its APIs. An account ID and Auth Token are obtained from the Twilio account, and this is used in the Action Manager to send smses to a selected number. Message to be sent is the login link based on the name given. Below is the Action that makes the API call:

```
class ActionSendToken(Action):
      def name(self):
            return "action_send_token"

      def run(self, dispatcher, tracker, domain):
            mydb =
mysql.connector.connect(host="35.166.18.143",user="emmanuel.annan"
,passwd="emmanuel.annan",database="webtech_emmanuel_annan")
        mycursor = mydb.cursor()
        f_name = tracker.get_slot('name')
        l_name = tracker.get_slot('last_name')
        sql = "SELECT * FROM students WHERE first_name = %s AND
last_name = %s"
        adr = (f_name,l_name)
        mycursor.execute(sql,adr)
      myresult = mycursor.fetchall()
            if len(myresult) == 0:
                student = "Sorry " + f_name + " " + l_name +" you
are not in our system"
            else:
                  account_sid =
"AC76cd680bb5124eda66ee5bbb80303c65"
                  auth_token = "401bafdcb494987ec8e83b44ae622f7d"
                  client = Client(account_sid, auth_token)

                  message = client.messages \
                              .create(
                                    body="Your login token is: " +
myresult[0][4]] + "\nKeep the token safe!",
                                    from_='+13475234873',
                                    to=myresult[0][3]
                               )
                  student = "Your login URL has been sent to your
phone number. Check and let me know if this was helpful"
                  dispatcher.utter_message(student)
                  return []
```

SMTP Library Module

The emails are sent using the smtplib module in python. The smptp server used is from Gmail. The module is supplied the destination address, subject, message and header information which would be used to send the email. It uses the sending email address to login and a generated password for the application to have permission to send emails. The Action to send an email can be found below:

```
 def sendemail(from_addr, to_addr_list, cc_addr_list, subject,
message, login, password,smtpserver='smtp.gmail.com:587'):
    header  = 'From: %s\n' % from_addr
    header += 'To: %s\n' % ','.join(to_addr_list)
    header += 'Cc: %s\n' % ','.join(cc_addr_list)
    header += 'Subject: %s\n\n' % subject
    message = header + message

    server = smtplib.SMTP(smtpserver)
    server.starttls()
    server.login(login,password)
    problems = server.sendmail(from_addr, to_addr_list, message)
    server.quit()
    return problems
```

This action implementation does not require information from RASA; hence it is not written with Python classes like previous actions. The email is information obtained from the log file that is written as a conversation occurs and is the only required part of the email.

Database connection

An SQL database modeled after the data on Chalkboard's actual database is hosted on a live server. Every Action that requires interacting with a database connects to this live server before executing the necessary SQL statements. The connection is established as:

```
mydb = mysql.connector.connect(host=[live server
address],user=[username],passwd=[password],database=[database
name])
mycursor = mydb.cursor()
```

Chapter 5: Testing and Evaluation

This chapter considers the implementation and whether it met the requirements specified earlier in the document. The below table lists the requirement to be tested, how it was tested, the outcome of the test and verdict. User testing was carried out with ten users. All percentage breakdowns of user feedback are available in the Appendix.

Requirement Specification	**Test Parameters and Specifications**	**Type of test**	**Outcome**
Problem Resolution Testing: Testing how well the bot solves users problems			
The bot should be able to resolve login issues	80% of users tested should have their login issues resolved	User Testing: Users indicated Yes, No or Yes with a few tries when asked if the bot resolved their login issues	100% of users tested indicated they had their login issues resolved. 50% indicated it was resolved outright while the other 50% indicated it took several tries to solve it
The bot should be able to recognize registered users of Chalkboard	The bot should be able to confirm the identities of 80% of tested users accurately	User Testing: Users indicated Yes, No or Yes with a few tries when asked if the bot could recognize them as users of Chalkboard or not	50% of users indicated that the bot was able to tell if they were a registered user or not outright and 30% were able to after several tries. 20% indicate the bot could not recognize them.
The bot should be able to solve enrolment issues	80% of users should have their enrolment issues resolved	User Testing: Users indicated Yes, No or Yes with a few tries when asked if the bot resolved their enrolment issues	80% of users indicated that the bot was able to resolve enrolment issues. 50% indicated that it was resolved outright while 30% indicated it took several tries. 20% indicated that the issue was not resolved at all
The bot should be able to solve	100% of users should be able to access	User Testing: Users indicated Yes, No or	100% of users indicated that the bot

navigation issues of users	resources to be able to learn how to use the application	Yes with a few tries when asked if the bot resolved their navigation issues	was able to resolve their navigation issue. 80% indicated that the issue was resolved outright and 20% indicated that it took several tries.
The bot should be able to resolve user queries in less time than its human counterpart	The bot should resolve all queries in the Issue base times table (Appendix 2.1) in no more time than it takes the human personnel	User and System Level Testing	100% if users indicated that when the bot resolved their issues, it took between 1 to 5 seconds to resolve it. The system was also tested without users and queries took less than 5 seconds to resolve.

5.1 Analysis of Problem Resolution Testing

The bot was successfully able to resolve the three main issues for which it was built. For some users, it took several tries for the bot to solve their problem. It was observed that the bot would have difficulty in extracting their name if their first name was a two-word name such as 'Paa Kofi' evident in Fig 1.1 in the Appendix. This resulted in the bot not being able to recognize 20% of users at some point in the conversation. In one unusual case, as shown in Fig 1.2 in the Appendix, when the user said 'I am Dennis Owusu,' the bot easily identifies the name but when the query is phrased as 'My name is Dennis Owusu,' it is unable to. Another user said 'My name is Kwasi Korboe' and the bot was able to extract the name in that instance. Once the bot was able to receive the name, it was successfully able to execute the follow-up action which was to solve enrollment, log in or navigation issues if the user was registered with Chalkboard and reject them if they were not. The rejection can be seen in Fig 1.3. The bot being unable to identify the names resulted in 50% of the users needing to make the query multiple times for Login problems and 30% for enrolment. 20% of users making enrollment queries did not have their issues resolved. The enrolment issue had another layer of difficulty because, beyond the name of the user, the bot also

required the names of the courses for which the student is enrolling. The available courses can be identified by course code, course name or both. Below are some of the courses:

JEN 123 - ENGLISH WITH ELEMENTS OF LITERATURE

JFL 127 - LITERATURE IN FRENCH II

JES 445 - FRENCH AS A FIRST LANGUAGE

JES 322 - DATA STRUCTURES

JES 128 - ENVIRONMENTAL AND SOCIAL STUDIES I

When a user tried to identify a course using course codes multiple times, the user was unable to enroll because the bot identified the wrong course. When another user identified the course by writing part of its name, the user was also unable to enroll because the bot once again identified the wrong course. The users who succeeded in getting enrolled used the full course name combined with the course code. The user who had to try multiple times attempted to register for multiple courses at once, and the bot could not identify the multiple courses. Hence they submitted singular queries after. 100% of users indicated that once the bot received the necessary information for a query, they received a response within 1 to 5 seconds. This is very important to the effectiveness of the dialog system as from the interview with the Chalkboard Operations Associate; the chatbot had to perform support processes much faster than the base time human personnel took. These base times can be found in Table 1.1 in the Appendix. By comparison, we can see that the bot performing in 1 to 5 seconds is a good improvement on the human personnel. Fig 1.5 in the Appendix shows a successful enrollment resolution.

Error handling and conversation logging testing: Testing how the well the bot can handle errors such as mistakes in query text and queries for which it does not know how to resolve as well as its ability to write logs of conversation for future learning			
The system should be able to learn from	The system should write and store a log	Component level testing	The system writes, and stores log files of

real-time queries to be more capable of handling queries it has not been trained on	file of user conversations for 100% of all tested conversations		conversations for all 100% conversations tested
The bot should be able to identify when it cannot solve a problem and refer to personnel	The bot should be able to send an email to Chalkboard personnel about a query it could not resolve only after trying at least once. 100% of times it cannot resolve a query it should report that fact back to the user and send the appropriate email	System Level and User Testing	80% of users indicated that the bot was able to tell them it was referring their problem to personnel when it could not handle it. 60% said it was able to do so right away and 20% said it could go after multiple tries. 20% of respondents indicated that the bot did not tell them it was referring to them to personnel
The bot should be able to work through simple spelling mistakes	The bot should be able to generate a response for the same ten sentences with at spelling errors in at least two words	System-level Testing	90% percent of users indicated that the bot was able to give useful responses when they made spelling mistakes. Users made deliberate spelling mistakes in repeated queries, and the bot was able to respond to their queries

5.2 Analysis of Error handling and conversation logging testing

The dialog system being able to write a log of the conversation was very crucial to its effectiveness. The logs serve as a way to automatically record user utterances for future use in retraining and diagnosing faults. That component of the system can do this regardless of its ability to generate the expected response. Fig 1.6 in the Appendix shows the folder on the local testing environment which stores the logs of conversation in text form. The bot was able to identify that it cannot handle a query and email personnel for 80% of the users. This 80% comprises of 60% who were told right away and 20% who were able to after

several tries. It was observed the 20% who needed multiple attempts, and the 20% who did not get referred at all met that change when they tried to resolve an enrollment issue. The bot kept extracting the wrong course name from the course names given due to the similarity in the course codes. Hence it would enroll the student for the wrong course and not identify that there is a problem. 90% of users indicated that the bot was able to work through spelling mistakes in similar queries. The users tested this by increasingly varying letters in words in their queries. The bot was able to maintain accuracy for four-letter changes in different words after which it generated inaccurate responses. Users used a maximum of 7 words in their queries; hence four letter changes are spread across the seven words. Fig 1.7, 1.8 and 1.9 in the Appendix shows an instance of a user incorporating spelling mistakes in its query.

User Experience testing: Testing the chatbot's naturalness, ability to generate intelligible responses and flexibility.			
The system should be able to process queries that have the same idea but appear in different forms and contexts.	The system should be able to generate accurate responses for three variations of a query given by users	System Level Testing and User Testing; System-Level Testing was done by inputting three variations of a query and observing the response. User testing saw users rate on a Linkert scale of 1 to 5 how well it was able to maintain accurate responses varying forms of a query were given	40% of users indicated that the bot was able to maintain accurate responses when they varied the form of the same query. 40% indicated that it was okay and 20% of users indicated that it did this badly.
The system should be able to respond to a few non-problem oriented queries to maintain some level of interactivity with the user.	The bot should be able to generate responses for up to 6 'off-topic' queries (small talk)	System Level Testing	The bot can generate responses for 8 'off-topic' queries
Talking to the bot should feel like talking to a human being	70% of tested users should give a score of 7 or higher on a Likert scale to	User Testing: Users are asked to rate on a Linkert scale from 1 to 5 where 1	70% of tested users indicated that the conversation with the bot felt quite human

	indicate how human the conversation felt	represents very robot like and five represents very human	with 50% giving a score of 4 and 20% giving a score of 5. 30% indicated it felt quite robotic, giving a score of 2.
The bot should be able to respond to user queries in under 20 seconds	Bot response time should be less than 20 seconds	User Testing: Users were asked to indicate how long the bot generally took to give a response with ranges of time in seconds given.	The bot sends responses to queries in less than 5 seconds. 90% of users indicated that across the bot to between 1 and 5 seconds and 10% indicated that it took between 5 and 10 seconds.
Users should be able to type in any problems they have or queries they need to be answered	Facebook Messenger platform should be online and Chalkboard page set-up to receive inputs	Component Testing	The Facebook Messenger platform connection was successful and was able to run throughout the testing period
The user should be able to read the generated response. The response should be in a form easily understandable by the user.	At least 80% of tested users should be able to attest that they understood system responses or 100% of 10 test queries should produce intelligible responses	User Testing	80% of users indicated that the bot was able to maintain accurate responses when they varied the form of the same query.

5.3 Analysis of User Experience Testing

The result of testing the bots ability to maintain accurate responses when the same query structure was varied can be described as mixed. 80% rated 3 and 4, and 20% rated 2. No user said it was able to maintain accuracy very well or very badly. Fig 2.1 in the Appendix shows an instance of the bot responding to varied queries. A potential explanation of this is the lack of training data with some intents. Some intents have about 7 to 15 sentences defined and hence would not make for a robust model for determining and classifying intents. It would identify some and miss others. There were also some out-of-vocabulary

words in some query variations and hence the trained model would not have any word embeddings for these words. Potential solutions to this would be discussed in the next chapter. The bot can generate responses to non-problem oriented queries. As mentioned earlier, small talk intents were added to the training data, and these small talk queries form part of the queries users varied and tested earlier. With regards to testing how human a conversation with the bot felt, 30% indicated it felt robotic with a Linkert scale of 2 whiles 50% and 20% gave it a score of 4 and five respectively. I observed that some users attempted to ask to follow up questions to the small talk intents for which the bot has not been designed to handle. It would respond with a generic greeting instead. The bots general response time was successfully less than 20 seconds. 100% of users indicated the bot took less than 20 seconds to respond. One user suggested that the bot took between 5 and 10 seconds to respond and it was observed that this was due to a lag in the internet connection that delayed the query being sent and response being delivered. Finally, the ability of the bot to respond well even when spelling mistakes were included also passes the test of its ability to generate an intelligible response when it receives a query.

Security Testing			
The bot should not give out sensitive information during the conversation	The bot should never reveal user log in the link directly but should be able to resolve the issue still	Component Level Testing	The bot successfully uses the Twilio API to send an SMS with the link to the user's phone number thereby preventing direct access of login links

5.3 Analysis of Security Testing

Security was a key concern in using the bot. The bot needed to avoid revealing key information directly but still being useful to users. The bot is successfully able to send a

request to the Twilio API to send an SMS to the user's phone number to avoid revealing student details. Fig 2.2 Shows the text login message sent from Twilio

Chapter 6: Conclusions and Recommendations

The bot can solve the three main problems of login, navigation, and enrollment. Its ability is hampered slightly by its ability to extract entities from diverse query structures. If it was able to, it successfully executed the necessary actions to resolve the problem. This means that improvement has to be made on the bots entity extraction to enable to identify entities in a query better. With this, users would not have to ask multiple times to have a problem resolved, and those who did not have it resolved altogether would not be in the same situation. Enrollment, in particular, has an added layer of difficulty because beyond needing the user name the bot needed the course name. The structure of the course names as a combination of similar course codes and course names made it difficult for the bot to distinguish between the different course names. This was primarily a problem when users sought to use shortened versions of course names instead of the full thing. A recommendation for this is to define synonyms in the training data that consist of shortened versions of longer course names. The ner_synonyms component in RASA can help define these synonyms to make the bot better at identifying course name entities. In this way, if a user uses a shorter version of the course name the bot would be able to identify it as a synonym and fill the appropriate slot with the full data. The bot can also omit course codes when listing courses and only show course names to encourage the user to use course names. The bot can also ask the user to affirm his choice of course name when the course codes are ambiguous.

Another recommendation, mainly geared towards addressing name extraction, would be to add more training data featuring more names existing in Chalkboards users. This would be useful since ultimately the system would be used primarily by students on Chalkboard's platform. In this way, the bot would be able to identify more easily the names of users, and repeated attempts won't need to be made to solve a problem. This would also

help the bot maintain accurate responses when a query is varied. More data on the small talk can be added to improve the humanity of the bot. This includes possible follow up questions to the small talk queries.

A recommendation for better handling out-of-vocabulary words and spelling mistakes would be to use language models trained on a larger corpus to give the bots model word embedding for a more significant number of words. The accuracy of the bot can be increased by ensuring that there are a balanced number of training examples per intent. The intents specified for the Chalkboard bot were not balanced as intents like the login intent had 25 samples and the hobby intent had seven examples. A lack of balance in training data can result in a biased classifier which can affect its accuracy negatively. Hence the recommendation here is to ensure balance in training examples

In terms of speed and efficiency, the bot can respond to queries and execute actions in less than 20 seconds as targeted. It must be noted however that it is at the mercy of internet speed hence various ways to improve speed times in future work are welcome.

All in all, I believe the project meets functional requirements to a suitable degree as the bot has the proven ability to resolve the three main problems outlined by the Chalkboard staff. The inaccuracies of the bot are helped by its ability to write and persist log data for future training. The bot is far from perfect, and the ability of the bot to successfully write logs is instrumental in getting it there. Its ability to also refer problems to personnel deserves to be highlighted as it enables demanding user queries to be still resolved by being brought to the attention of human personnel while solving the simpler ones. In this way, it reduces the amount of work the Operations staff to perform and frees them to perform other tasks.

6.1 Future Work

Considering the prevalence of USSD applications in West Africa (find source), the chatbot can also be made accessible through a USSD application to enable it to be used without internet access. More training sentences and stories should be added for the chatbot to be trained on to make it even more conversational and human. It should also be tested with an even higher number of users to capture new intents, sample sentences, and conversations that can help to improve the bot. The recommendations highlighted above should also be implemented to make the bot more effective.

Appendices

Table 1.1: Showing the base times taken for problem resolution at Chalkboard

Information Obtained from Interview with Paa Kofi Antwi Larbi, an Operations Associate at Chalkboard Education			
Issue Type	**Reasons**	**Estimated Base Time (Best Time)**	**Steps**
Login Issue	Misplaced URL	120 seconds	Go into API
			Search for client name with CTRL+F
			Check box by name
			Click Send URL button
Navigation Issue	Not knowing how to move through the app	10 seconds	Refer to manual by sending link
Content Issue	Cannot access course/Course not showing	150 seconds	Go into API
			Navigate to institution courses
			Select the course client says is not showing.
			Navigate to students enrolled in that course
			Check client name
			Click assign button
			Ask the client to refresh.

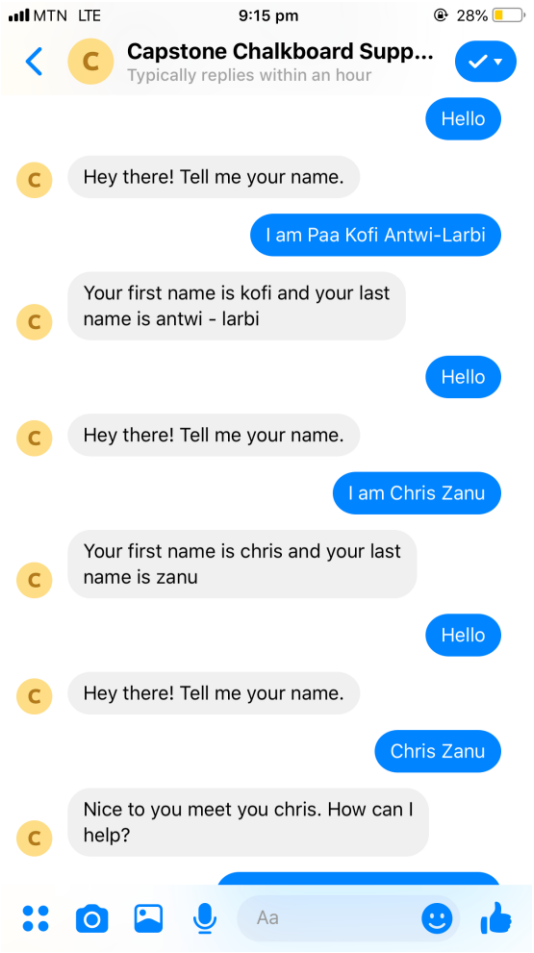

MTN LTE
9:15 pm
28%
Capstone Chalkboard Supp...
Typically replies within an hour
Hello
Hey there! Tell me your name.
I am Paa Kofi Antwi-Larbi
Your first name is kofi and your last name is antwi - larbi
Hello
Hey there! Tell me your name.
I am Chris Zanu
Your first name is chris and your last name is zanu
Hello
Hey there! Tell me your name.
Chris Zanu
Nice to you meet you chris. How can I help?
Aa

Fig 1.1

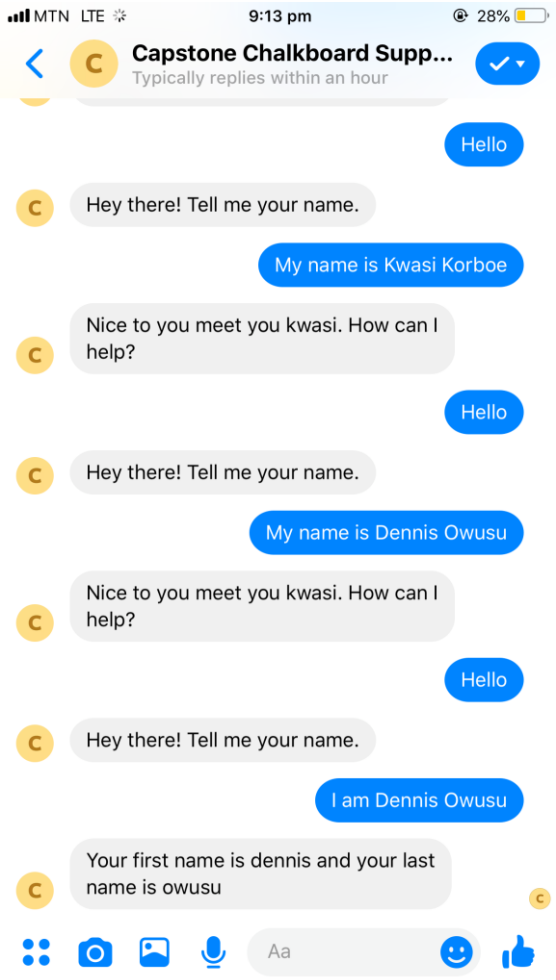

MTN LTE
9:13 pm
28%
Capstone Chalkboard Supp...
Typically replies within an hour
Hello
Hey there! Tell me your name.
My name is Kwasi Korboe
Nice to you meet you kwasi. How can I help?
Hello
Hey there! Tell me your name.
My name is Dennis Owusu
Nice to you meet you kwasi. How can I help?
Hello
Hey there! Tell me your name.
I am Dennis Owusu
Your first name is dennis and your last name is owusu
Aa

Fig 1.2

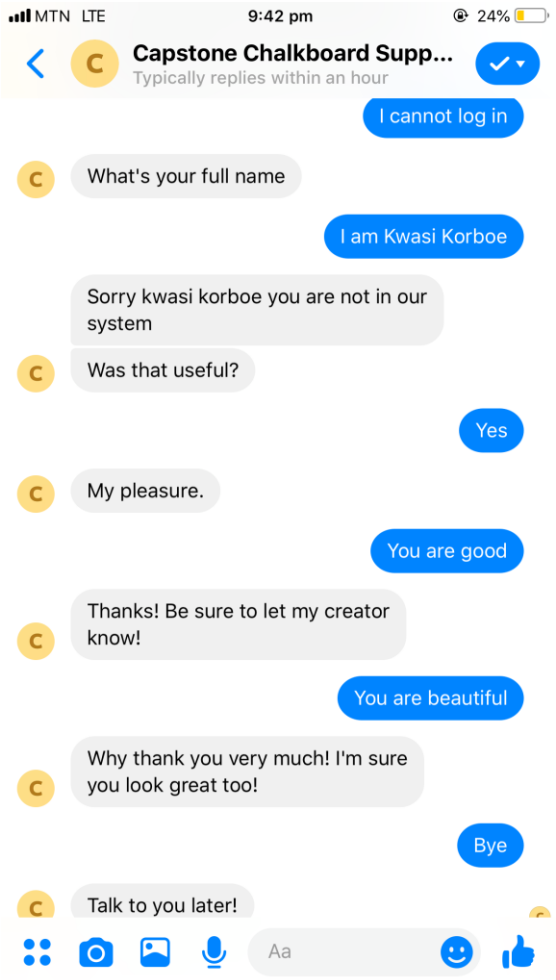

MTN LTE
9:42 pm
24%
Capstone Chalkboard Supp...
Typically replies within an hour
I cannot log in
What's your full name
I am Kwasi Korboe
Sorry kwasi korboe you are not in our system
Was that useful?
Yes
My pleasure.
You are good
Thanks! Be sure to let my creator know!
You are beautiful
Why thank you very much! I'm sure you look great too!
Bye
Talk to you later!
Aa

Fig 1.3

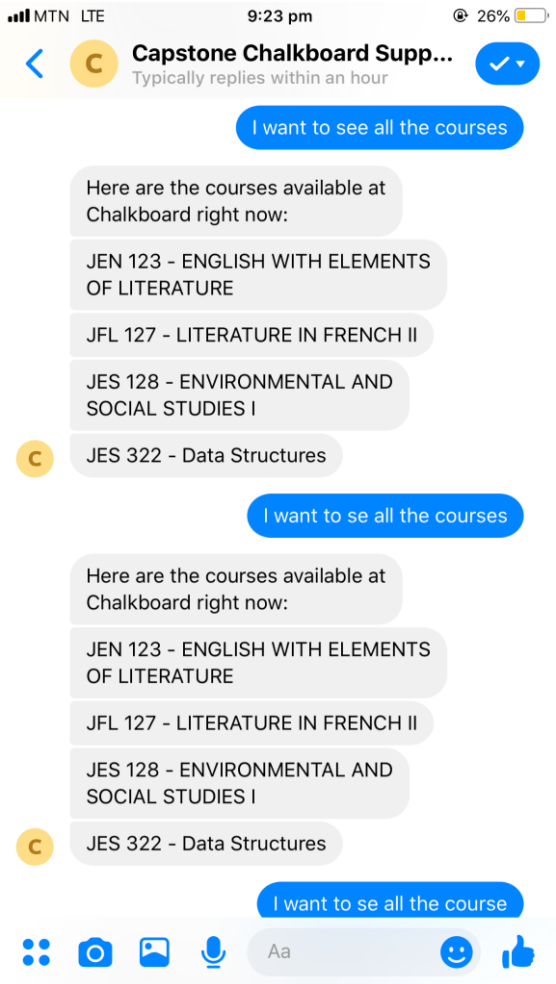

MTN LTE
9:23 pm
26%
Capstone Chalkboard Supp...
Typically replies within an hour
I want to see all the courses
Here are the courses available at Chalkboard right now:
JEN 123 - ENGLISH WITH ELEMENTS OF LITERATURE
JFL 127 - LITERATURE IN FRENCH II
JES 128 - ENVIRONMENTAL AND SOCIAL STUDIES I
JES 322 - Data Structures
I want to se all the courses
Here are the courses available at Chalkboard right now:
JEN 123 - ENGLISH WITH ELEMENTS OF LITERATURE
JFL 127 - LITERATURE IN FRENCH II
JES 128 - ENVIRONMENTAL AND SOCIAL STUDIES I
JES 322 - Data Structures
I want to se all the course
Aa

Fig 1.4

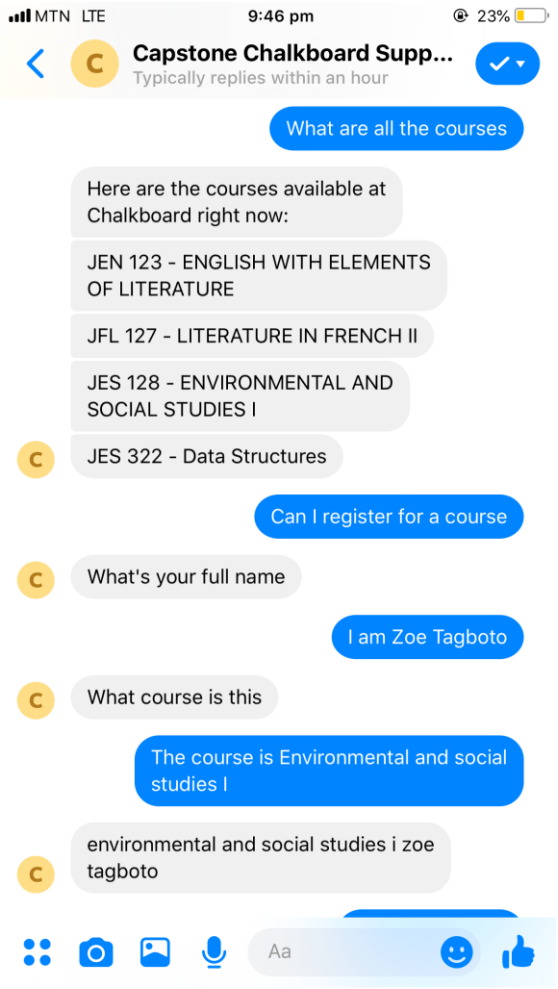

MTN LTE
9:46 pm
23%
Capstone Chalkboard Supp...
Typically replies within an hour
What are all the courses
Here are the courses available at Chalkboard right now:
JEN 123 - ENGLISH WITH ELEMENTS OF LITERATURE
JFL 127 - LITERATURE IN FRENCH II
JES 128 - ENVIRONMENTAL AND SOCIAL STUDIES I
JES 322 - Data Structures
Can I register for a course
What's your full name
I am Zoe Tagboto
What course is this
The course is Environmental and social studies I
environmental and social studies i zoe tagboto
Aa

Fig 1.5

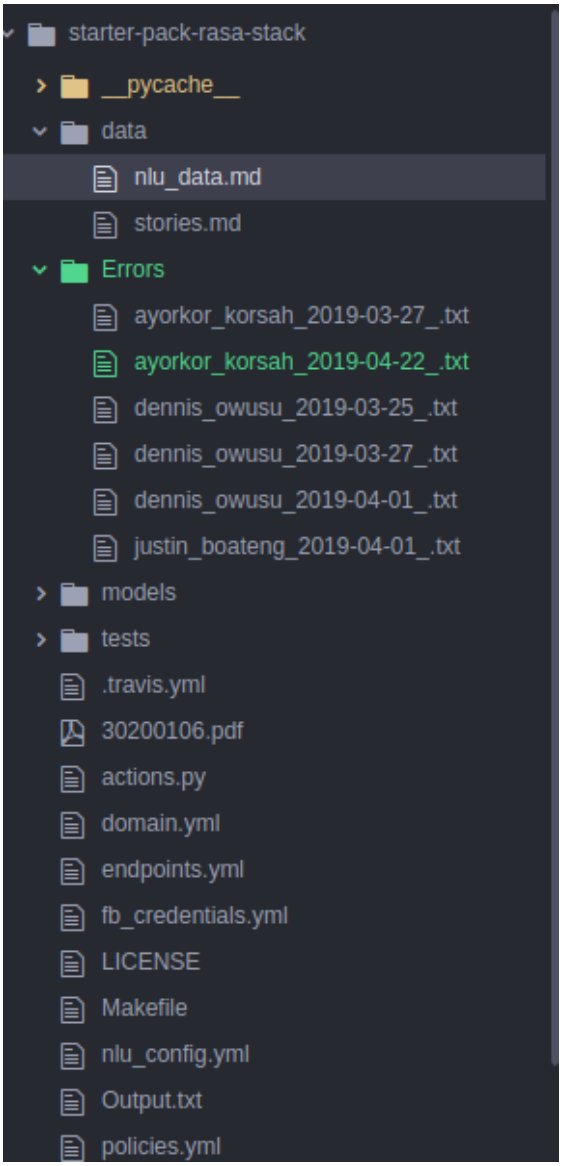

starter-pack-rasa-stack
__pycache__
data
nlu_data.md
stories.md
Errors
ayorkor_korsah_2019-03-27_.txt
ayorkor_korsah_2019-04-22_.txt
dennis_owusu_2019-03-25_.txt
dennis_owusu_2019-03-27_.txt
dennis_owusu_2019-04-01_.txt
justin_boateng_2019-04-01_.txt
models
tests
.travis.yml
30200106.pdf
actions.py
domain.yml
endpoints.yml
fb_credentials.yml
LICENSE
Makefile
nlu_config.yml
Output.txt
policies.yml

Fig 1.6

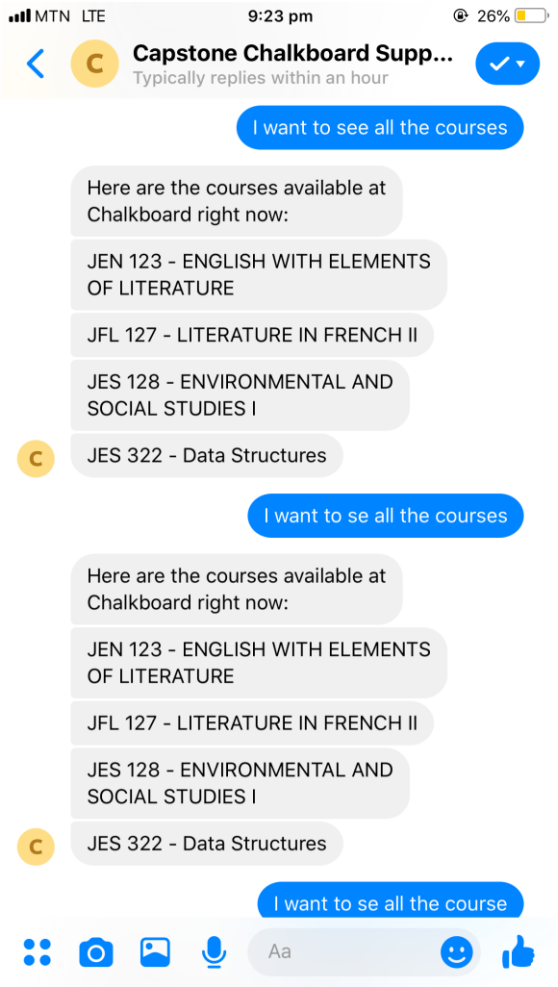

MTN LTE
9:23 pm
26%
Capstone Chalkboard Supp...
Typically replies within an hour
I want to see all the courses
Here are the courses available at Chalkboard right now:
JEN 123 - ENGLISH WITH ELEMENTS OF LITERATURE
JFL 127 - LITERATURE IN FRENCH II
JES 128 - ENVIRONMENTAL AND SOCIAL STUDIES I
JES 322 - Data Structures
I want to se all the courses
Here are the courses available at Chalkboard right now:
JEN 123 - ENGLISH WITH ELEMENTS OF LITERATURE
JFL 127 - LITERATURE IN FRENCH II
JES 128 - ENVIRONMENTAL AND SOCIAL STUDIES I
JES 322 - Data Structures
I want to se all the course
Aa

Fig 1.7

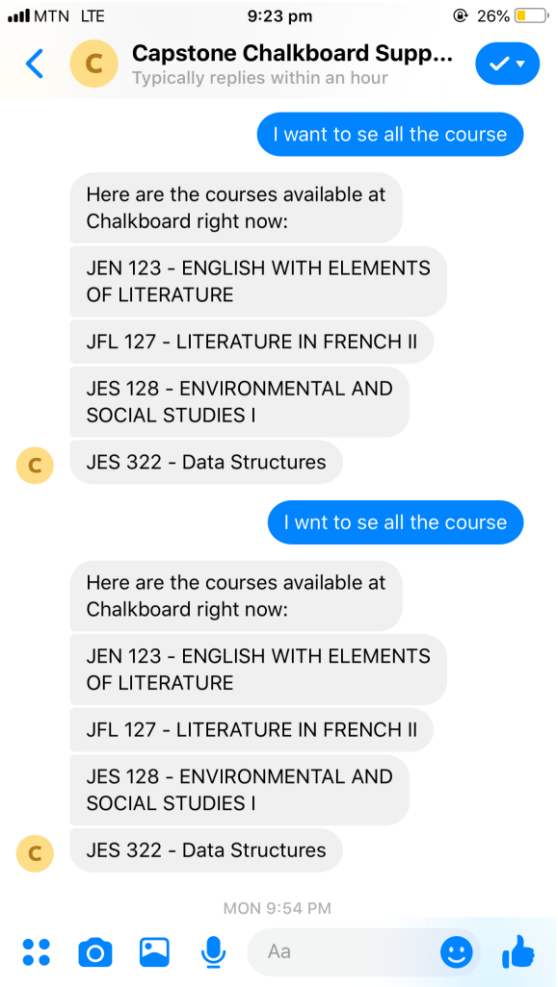

MTN LTE
9:23 pm
26%
Capstone Chalkboard Supp...
Typically replies within an hour
I want to se all the course
Here are the courses available at Chalkboard right now:
JEN 123 - ENGLISH WITH ELEMENTS OF LITERATURE
JFL 127 - LITERATURE IN FRENCH II
JES 128 - ENVIRONMENTAL AND SOCIAL STUDIES I
JES 322 - Data Structures
I wnt to se all the course
Here are the courses available at Chalkboard right now:
JEN 123 - ENGLISH WITH ELEMENTS OF LITERATURE
JFL 127 - LITERATURE IN FRENCH II
JES 128 - ENVIRONMENTAL AND SOCIAL STUDIES I
JES 322 - Data Structures
MON 9:54 PM
Aa

Fig 1.8

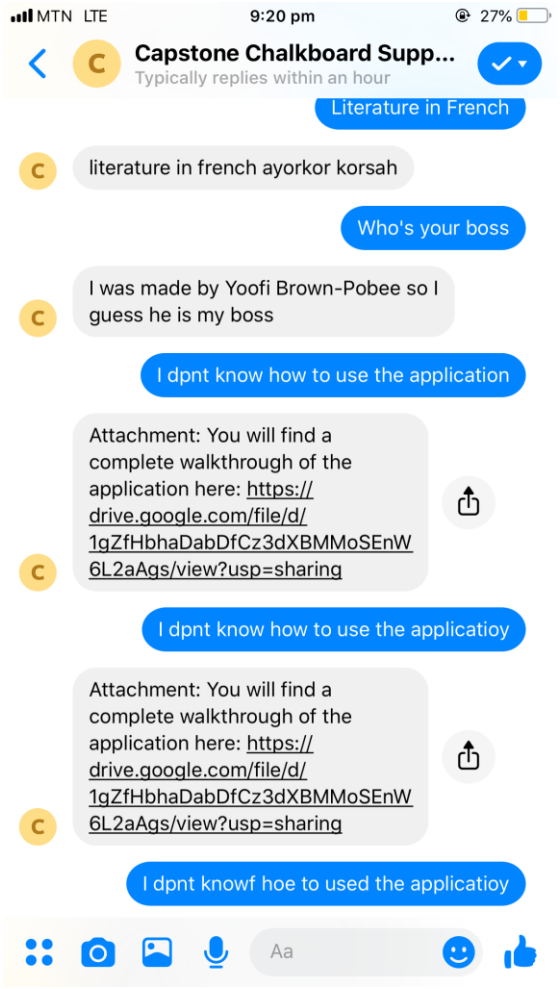

MTN LTE
9:20 pm
27%
Capstone Chalkboard Supp...
Typically replies within an hour
Literature in French
literature in french ayorkor korsah
Who's your boss
I was made by Yoofi Brown-Pobee so I guess he is my boss
I dpnt know how to use the application
Attachment: You will find a complete walkthrough of the application here: https://drive.google.com/file/d/1gZfHbhaDabDfCz3dXBMMoSEnW6L2aAgs/view?usp=sharing
I dpnt know how to use the applicatioy
Attachment: You will find a complete walkthrough of the application here: https://drive.google.com/file/d/1gZfHbhaDabDfCz3dXBMMoSEnW6L2aAgs/view?usp=sharing
I dpnt knowf hoe to used the applicatioy
Aa

Fig 1.9

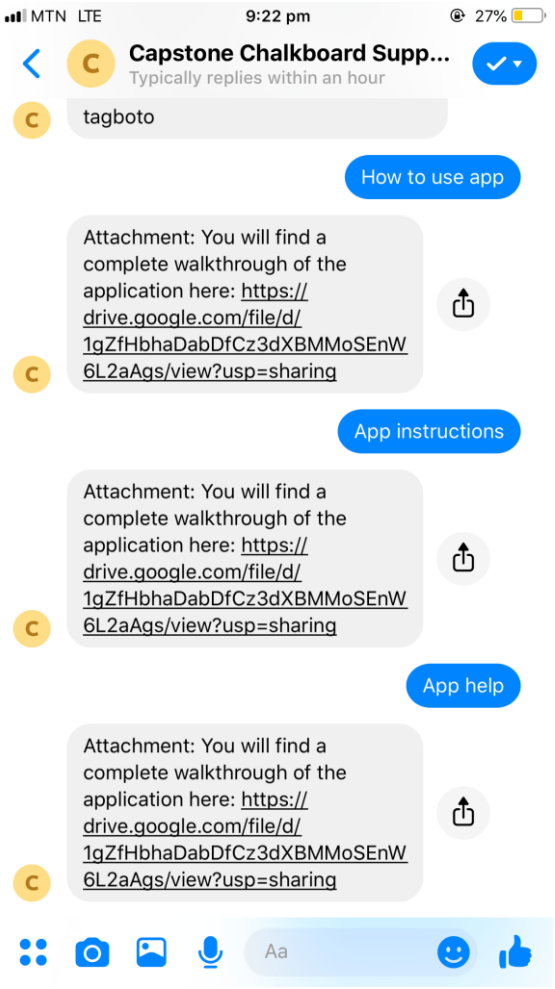

MTN LTE
9:22 pm
27%
Capstone Chalkboard Supp...
Typically replies within an hour
tagboto
How to use app
Attachment: You will find a complete walkthrough of the application here: https://drive.google.com/file/d/1gZfHbhaDabDfCz3dXBMMoSEnW6L2aAgs/view?usp=sharing
App instructions
Attachment: You will find a complete walkthrough of the application here: https://drive.google.com/file/d/1gZfHbhaDabDfCz3dXBMMoSEnW6L2aAgs/view?usp=sharing
App help
Attachment: You will find a complete walkthrough of the application here: https://drive.google.com/file/d/1gZfHbhaDabDfCz3dXBMMoSEnW6L2aAgs/view?usp=sharing
Aa

Fig 2.1

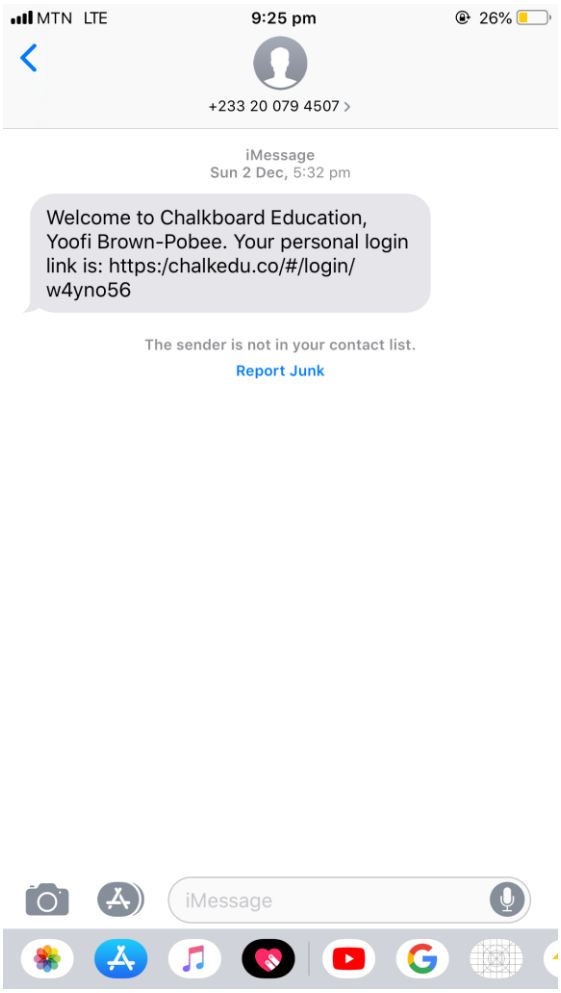


Fig 2.2

Full Bot Domain:
intents:
- greet
- goodbye
- thanks
- deny
- give_course
- joke
- give_name
- login
- navigation
- enrollment
- affirm
- list_course
- introduction
- age
- beautiful
- birthday
- boss

- help
- good
- hobby

entities:
- name
- last_name
- course_name

slots:
  name:
    type: text

  last_name:
    type: text

  course_name:
    type: text

actions:
- utter_name
- utter_thanks
- utter_greet
- utter_goodbye
- action_joke
- action_send_token
- utter_ask_for_name
- utter_full_name
- utter_useful
- utter_try_again
- utter_refer_to_person
- utter_ask_for_course
- action_enrol_student
- action_list_all_courses
- utter_send_manual
- action_send_manual
- action_write_log
- action_send_email
- utter_introduction_response
- utter_age_response
- utter_beautiful_response
- utter_birthday_response
- utter_boss_response
- utter_help_response
- utter_good_response
- utter_hobby_response

```
templates:
  utter_name:
  - text: "Hey there! Tell me your name."

  utter_greet:
  - text: "Nice to you meet you {name}. How can I help?"

  utter_goodbye:
  - text: "Talk to you later!"

  utter_thanks:
  - text: "My pleasure."

  utter_ask_for_name:
  - text: What's your full name

  utter_full_name:
  - text: "Your first name is {name} and your last name is {last_name}"

  utter_useful:
  - text: "Was that useful?"

  utter_try_again:
  - text: "Okay, {name} lets try this again"

  utter_refer_to_person:
  - text: "Looks like your issue is a bit more nuanced and I will forward it to Operations to
get in touch with you shortly"

  utter_restart:
  - text: "Anything else"

  utter_ask_for_course:
  - text: "What course is this"

  utter_send_manual:
  - text: "You will find a complete walkthrough of the application here:
https://drive.google.com/file/d/1gZfHbhaDabDfCz3dXBMMoSEnW6L2aAgs/view?usp=s
haring"

  utter_introduction_response:
  - text: "My name is Sally, and I work for Chalkboard. I love listening to your problems
and sending emails!"

  utter_age_response:
  - text: "I'm about two months old you know!"

  utter_beautiful_response:
  - text: "Why thank you very much! I'm sure you look great too!"
```

utter_birthday_response:
- text: "I was born on 3rd May 1996. Same as my creator!"

utter_boss_response:
- text: "I was made by Yoofi Brown-Pobee so I guess he is my boss"

utter_help_response:
- text: "I am here for you 24/7"

utter_good_response:
- text: "Thanks! Be sure to let my creator know!"

utter_hobby_response:
- text: "I like to sit here and wait till you have a problem!"

References

[1] Daniel Jurafsky and James H. Martin. 2014. *Speech and language processing*, Harlow: Pearson.

[2] Anon. What is Turing test? - Definition from WhatIs.com. Retrieved October 10, 2018 from https://searchenterpriseai.techtarget.com/definition/Turing-test

[3] Corydon Ireland. 2012. Alan Turing at 100. (September 2012). Retrieved October 10, 2018 from https://news.harvard.edu/gazette/story/2012/09/alan-turing-at-100/

[4]Robert Gehl. Teaching to the Turing Test with Cleverbot. *The Journal of Inclusive Scholarship and Pedagogy*24, 1-2.

[5]Carpenter. Cleverbot. Retrieved October 9, 2018 from http://www.cleverbot.com/

[6]Anjuli Kannan and Oriol Vinyals. 2017. Adversarial Evaluation of Dialogue Models. (2017).

[7]Ming Zhou, Lei Cui, Shaodan Huang, Furu Wei, Chuanqi Tan, and Chaoqun Duan. SuperAgent: A Customer Service Chatbot for E-commerce Websites. Retrieved April 5, 2019 from http://aclweb.org/anthology/P17-4017

[8]Carmen Holotescu. MOOCBuddy: a chatbot for personalized learning with MOOCs. Retrieved April 5, 2019 from

https://www.researchgate.net/publication/304037510_MOOCBuddy_a_chatbot_for_personalized_learning_with_MOOCs

[9]N.T. Thomas. 2016. An e-business chatbot using AIML and LSA. *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*(2016). DOI:http://dx.doi.org/10.1109/icacci.2016.7732476

[10]Tobias Wochinger. 2019. Rasa NLU in Depth: Part 1 - Intent Classification. (February 2019). Retrieved April 22, 2019 from https://medium.com/rasa-blog/rasa-nlu-in-depth-part-1-intent-classification-cb17e27fb169

[11]Anon. Choosing a Rasa NLU Pipeline. Retrieved April 22, 2019 from https://rasa.com/docs/nlu/choosing_pipeline/

[12]K.u. Senevirathne, N.s. Attanayake, A.w.m.h. Dhananjanie, W.a.s.u. Weragoda, A. Nugaliyadde, and S. Thelijjagoda. 2015. Conditional Random Fields based Named Entity Recognition for Sinhala. 2015 IEEE 10th International Conference on Industrial and Information Systems (ICIIS) (2015). DOI:http://dx.doi.org/10.1109/iciinfs.2015.7399028

[13]Charles Sutton and Andrew McCallum. 2012. An Introduction to Conditional Random Fields. Foundations and Trends® in Machine Learning 4, 4 (2012), 267–373. DOI:http://dx.doi.org/10.1561/2200000013

[14]Freddy Boulton. 2018. Conditional Random Field Tutorial in PyTorch . (May 2018). Retrieved April 22, 2019 from https://towardsdatascience.com/conditional-random-field-tutorial-in-pytorch-ca0d04499463

[15]Fabian Pedregosa et al.2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*(October 2011), 2825–2830.

[16]Anon. Keras: The Python Deep Learning library. Retrieved April 23, 2019 from https://keras.io/