
Application of Chatbots in Education

Svetlozar Georgiev -
40203970

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of
BEng (Hons) Software Engineering

School of Computing

April 3, 2019

Authorship Declaration

I, Svetlozar Georgiev Georgiev, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained informed consent from all people I have involved in the work in this dissertation following the School's ethical guidelines.

Signed:

Date: 03/04/2019

Matriculation no: 40203970

General Data Protection Regulation Declaration

Under the General Data Protection Regulation (GDPR) (EU) 2016/679, the University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below one of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Abstract

This report proposes a system which can be used in an educational environment to provide assistance to both students and teaching staff. Such systems, called chatbots, are used in fields such as marketing, automated customer support, entertainment, however they have not found an application in the area of education. The aim of the suggested software is to respond to questions of Computing students about programming, however the proposed system is designed with reusability in mind so that it can be modified to gain knowledge in any field.

The relevant background knowledge is acquired by reviewing academic literature on the relevant topics for chatbot development: Artificial Intelligence and its branches. Additionally, the history of chatbots is investigated so that a comparison between the approaches to their development can be made.

The development process of the proposed system is described. The finished prototype implements most of the initial objectives, however some problems are discovered during the evaluation stage. A group of users, which represent the target audience of the application, tests the prototype and completes a questionnaire. The results show that the users are satisfied with the system, however they highlight problems with it.

The conclusion is that the system has potential, however in its current form its usage is limited and for this reason, some suggestions for future improvements are provided.

Contents

1	Introduction	1
1.1	Background	1
1.2	Aim and Objectives	1
1.3	Scope and Constraints	2
1.4	Structure	2
2	Literature Review	3
2.1	Chatbots	3
2.1.1	The Turing Test	3
2.1.2	Evolution of Chatbots	5
2.2	Artificial Intelligence	7
2.2.1	Definition	7
2.3	Machine Learning	8
2.3.1	Supervised Learning	9
2.3.2	Unsupervised Learning	11
2.3.3	Semi-supervised Learning	11
2.3.4	Reinforcement Learning	12
2.4	Natural Language Processing	12
2.4.1	Morphological	13
2.4.2	Syntactic	13
2.4.3	Semantic	15
2.4.4	Approaches to NLP	16
2.5	Summary	17
3	Technology Review	18
3.1	Chatbot Library	18
3.2	Chatterbot	19
3.2.1	Overview of Input Processing	19
3.2.2	Configuration	19
3.2.3	Preprocessors	19
3.2.4	Logic Adapters	19
3.2.5	Storage Adapters	23
3.2.6	Training	23
3.2.7	Generating a response	24
3.3	Python	24
3.4	Flask	25
3.4.1	Twitter Bootstrap	26
3.5	Pytest	27

3.6	Web Hosting	27
4	Implementation	28
4.1	DevOps	28
4.1.1	Management	28
4.1.2	Version control	29
4.1.3	Testing	29
4.1.4	Code Coverage	30
4.1.5	Continuous Integration & Deployment	31
4.2	Application	32
4.3	Back-end	33
4.3.1	Web Crawler	33
4.3.2	Chatbot	39
4.4	Front-end	40
4.4.1	Project Structure	40
4.4.2	Application Setup	41
4.4.3	Server-side Template Rendering	42
4.4.4	Client-side Template Rendering	44
4.5	Back-end and Front-end Communication	47
4.5.1	Requests	47
4.6	Extra Features	48
4.6.1	Command line Interface	48
4.6.2	Alternate Responses	49
4.6.3	Feedback	50
4.7	Summary	52
5	Evaluation	54
5.1	Preparation	54
5.2	Comparison with Cleverbot	55
5.3	User Testing	58
5.3.1	First Testing Session	59
5.3.2	Second Testing Session	61
6	Conclusions	69
7	Future Work	71
	References	72
	Appendices	75
	A Project Overview	75

B Initial Project Plan	78
C Diary Sheets	79
D Second Formal Review Output	95
E Results from the First Testing Session	97
F Survey Questions	98
G Web Crawling Program	100

List of Tables

1	Comparison of the results of stemming and lemmatisation of some words.	15
2	Parameters used for the web crawler when collecting data in preparation for testing.	55
3	A summary of the results from the questionnaire.	67
4	Results from the first testing session.	97
5	The questions each tester was asked in the survey and their possible answers.	98

List of Figures

1	The process of training a learning algorithm.	9
2	Example list of stopwords. Adapted from (Manning, Raghavan, & Schütze, 2008).	14
3	Diagram of the process of receiving input, processing it and generating a response (Gunther Cox, 2019).	20
4	Process flow diagram of how responses are selected	22
5	The table structure of the relational database used to store the “knowledge” of a chatbot.	24
6	An example of a prioritised sprint backlog.	28
7	The structure of a Stack Overflow URL.	34
8	Example training file in the YAML format.	38
9	A flow diagram of the data collection process.	39
10	The interface of the website.	43
11	A summary of request processing in the back-end.	53
12	The improved user interface.	61
13	Bar chart showing the overall programming knowledge level of the participants in the survey.	63
14	Bar chart showing the responses to question 2.	63
15	Bar chart showing the responses to question 5.	64
16	Bar chart showing the responses to question 6.	65
17	Bar chart showing the responses to question 9.	66

Listings

1	Example implementation of the <code>can_process()</code> method. Adapted from (Gunther Cox, 2019).	21
2	A code coverage report produced with <code>pytest-cov</code>	30
3	An example hyperlink to a Stack Overflow question.	35
4	Extracting answers for a specific Stack Overflow question.	36
5	Extracting the URL of each question on a Stack Overflow page.	37
6	Writing scraped data to a file in the YAML format.	38
7	Multithreaded optimisation of the web crawler.	38
8	The file structure of the project.	41
9	Initialisation of the home page of the front-end application.	42
10	Using a variable passed from Python in a Jinja template.	43
11	The HTML code used to create a chatbox message sent by the chatbot.	45
12	The code for the JsRender message template.	46
13	Rendering the JsRender message template.	46
14	Example data sent to the back-end for processing in the JSON format.	48
15	The JsRender template used to generate HTML code for a feedback panel.	51
16	The source code of the second web crawling program.	100

Acknowledgements

Firstly, I am extremely grateful to my supervisor, Dr Kevin Sim, for his help, support and great ideas without which I would not have been able to finish this project or write this dissertation.

In addition, I would like to thank my family for giving me the opportunity to pursue a higher education and supporting me throughout it.

I wish to also express my gratitude to all my friends who provided me with much needed distractions and encouragement.

1 Introduction

The aim of this project is to investigate whether it is viable to use chatbots within an educational context.

1.1 Background

Since the idea of intelligent machines was first suggested, there has been an ongoing interest in developing a machine that can communicate with natural language (such as English). This is no surprise since one of the definitive qualities of humans is their desire for social interaction. However, interacting with computers is achieved in a language so difficult to understand that it is referred to as “code”. As a result, programs called chatbots have been developed to allow people to communicate with machines. The initial reason for implementing them was for entertainment purposes, however, presently such systems see practical usage in different areas: marketing, information retrieval, customer service, etc. However, one area where they have not been implemented is education.

Learning is usually a process which involves a teacher and it is achieved through exchange of information with natural language. However, learning programming has proven to be a difficult task. As a result, new university students often struggle to grasp some of its concepts. For this reason, this report proposes a system which, given knowledge about a subject matter, can help both lecturers and students. The former will be relieved of the need to answer frequently asked questions, while the latter will be able to receive an immediate response to their query.

1.2 Aim and Objectives

The aim of this project is to develop and evaluate a “smart” chatbot which can be applied in an educational environment. That is, it will be able to assist Computing students at university who are new to programming and are struggling to solve a specific problem. Furthermore, it will aid teaching staff in answering recurring questions. The goals for the deliverable are:

- A graphical user interface which will allow the user to interact with the system via textual input.
 - The graphical user interface will be easy to understand and use.
 - The system will be able to use Artificial Intelligence techniques to analyse its input.
-

- The system will be able to use Artificial Intelligence techniques to generate an appropriate response user enquiries.
- The system will be able to improve through its interaction with its users.
- The system will be accessible from all types of devices which support Internet connectivity.
- The members of the teaching staff who wish to use the system will be able to train it easily with information about anything they see fit.

1.3 Scope and Constraints

This project will not attempt to develop a dedicated Artificial Intelligence system. Instead, a ready-made solution will be used in the form of a library for a programming language or a service.

1.4 Structure

This report consists of the following chapters:

1. **Introduction** — introduces the topic and background of the project, defines the deliverable, defines the constraints and explains the structure of the report.
 2. **Literature Review** — presents a summary of available academic literature related to the concepts and technologies required for the development of this project. The topics covered are: chatbots and a brief history of their evolution; Artificial Intelligence and its branches relevant to the development of chatbots: Machine Learning and Natural Language Processing.
 3. **Technology Review** — illustrates the technologies and methods used to develop the deliverable.
 4. **Implementation** — describes how the application was developed.
 5. **Evaluation** — compares the functionality of the produced system with the initially set goals. Additionally, user testing results are summarised.
 6. **Conclusion** — summarises how well the project was developed and whether the objectives were met.
 7. **Future Work** — describes the areas of in which the project did not meet the initial plan and provides suggestions for improvements.
-

2 Literature Review

This chapter discusses the underlying techniques and required background knowledge for the development of a chatbot system.

2.1 Chatbots

Michael L. Mauldin (1994) was the first to use the term *ChatterBot* from which the modern term *chatbot* was derived. He defined it as *a computer program which conducts a textual conversation with its user* (Michael L. Mauldin, 1994). The aim of such programs is to simulate a human-like ability to communicate intelligently with natural language.

To better explain how chatbots operate, it would be necessary to introduce Artificial Intelligence (AI) and its relevant branches. What follows is a history of AI.

2.1.1 The Turing Test

The concept of intelligent computers was first introduced by Alan Turing in his paper *Computing Machinery and Intelligence* (Turing, 1950). He predicted that by the 2000s, computers with such capacity would be able to execute programs that will be able to communicate with their users and convince them that they are talking to another human 70% of the time. In addition, Turing presents a method of judging whether a machine is able to exhibit intelligent behaviour similar to that of a human: *the Turing test*.

There three participants in the Turing test: a computer program which is being evaluated, a human participant and a human judge. They are separated so that the judge does not know who they are communicating with. Instead, the communication is achieved through textual input via a computer. The judge asks both participants questions and tries to guess which one is the computer program. This process is repeated a set amount of times. It is then counted how many times the evaluator thought that the machine was the human participant. If it is 50% of the times or more, the machine can be considered to have passed the Turing test and communicated intelligently.

Alan Turing's ideas provoked great interest in the research of intelligent machines. A competition called "the Loebner Prize" was created which utilised Turing's method of judging the intelligence of machines.

The Loebner Prize

The Loebner Prize is an annual competition created in 1990 by Hugh Loebner where computer programs are judged on their ability to understand language (Loebner.net, 2017). The participating programs are tested with the Turing test. The winners of the competition are awarded monetary prizes.

The importance of this contest is that it has incentivised many people to develop chatbots. It can be said that it has contributed to the development of the field of AI by attracting more interest to it.

However, the contest has been widely criticised. For instance, Floridi, Taddeo, and Turilli (2009) describe the 2008 competition where judges asked unsuitable (“yes” or “no”) questions which could not effectively evaluate a machine’s ability to “think”. Additionally, conversations between the chatbots and judges were only 2.5 minutes of length. According to Floridi et al. (2009), the format of the competition also encouraged contestants to create bots which paraphrase and repeat the question when they couldn’t answer.

Other arguments against it target the method of evaluation, the Turing test. An example is John Searle’s “Chinese Room Argument” (Searle, 1980).

The Chinese Room Argument

In the 1980 paper *Minds, Brains and Programs*, Searle (1980) argues that the Turing test cannot be used to determine whether a machine is truly intelligent. He proposes a thought experiment where a computer program which is able to pass the Turing test has been created. The program is able to communicate fluently in Chinese as well as a native human speaker. Searle (1980) claims that he could pass the Turing test too by following the same instructions as the program. He argues that since the machine was programmed to process the input, it does not understand it. Therefore, it must be imitating its ability to think and it does not have a mind.

Summary

Turing’s research and ideas provoked interest in the concept of machine intelligence. He can be considered the creator of a new field in science: Artificial Intelligence. It can, however, be argued that the Turing test is not suitable method for evaluating intelligence. For instance, animals which are considered to exhibit intelligent behaviour such as dolphins, dogs, primates cannot pass the Turing test. This because the test does not take into account the ability of many chatbots to imitate intelligent communication.

2.1.2 Evolution of Chatbots

Chatbots have existed since the 1960s with the inception of ELIZA (Weizenbaum, 1966). The first chatbots were created for entertainment purposes. They utilised simple pattern-matching techniques to compare their input to known data. In the 1970s and 1980s, the interest of developers was shifted to studying natural language (Wilensky et al., 1988) and more elaborate programs were built. Since then, there has been constant research in the relevant scientific areas used for the creation of chatbots. A chronological list of popular chatbots is presented below to illustrate how such programs have been evolving throughout the years.

ELIZA

In 1966, Joseph Weizenbaum created the first public chatbot, ELIZA (Weizenbaum, 1966). It was the first widely recognised program able to pass the Turing Test. It allows its users to chat with a Rogerian psychologist¹. ELIZA employed a pattern-matching method to simulate a conversation. The input is paraphrased, matched to a list of known statements, and a response is selected from a list of known answers (Weizenbaum, 1966). As a result, the program had no understanding of context and could only answer separate questions, making it unable to conduct a meaningful conversation. Additionally, ELIZA was programmed to answer with generic responses when there is no known reply. The reason ELIZA was created was to show how superficial a conversation between a computer and human is, however it became widely popular because it managed to convince a lot of people that they were talking to a real person.

PARRY

PARRY was created in 1972 by the psychiatrist Kenneth Kolby. Its aim was to simulate a paranoid schizophrenic. It used a similar rule-based system to ELIZA (Colby, 1981).

Jabberwacky

Jabberwacky was initially created by the British programmer Rollo Carpenter in 1981 and publicly launched in 1997 (Carpenter, 1997). The main purpose for the creation of the program was to pass the Turing Test. It was one of the first chatbots which utilised AI to learn from conversations with its users. It could learn facts, concepts and new languages. Instead of using a rule-based system, it relied on feedback and context. In 2004, it was awarded a second place in the Loebner Prize competition.

¹Rogerian psychology is a person-centered approach to psychology where the psychologist encourages the client to solve their problem themselves by asking them a series of questions.

A.L.I.C.E.

Artificial Linguistic Internet Computer Entity (A.L.I.C.E.) was developed by Dr. Richard Wallace in 1995. It was inspired by ELIZA and it utilised a similar rule-based method which could be improved with AIML (Artificial Intelligence Markup Language). The rules (called categories) followed by the program are defined in this language manually by a person. A consists of a question and an answer. The systems stores these rules in a tree structure managed by an object called the *graphmaster* which utilises a storage and matching algorithm (Wallace, 2009, p. 182). The graphmaster is well-optimised and allows for efficient speed of pattern matching.

SmarterChild

SmarterChild was initially launched in the early 2000s. Users could message it via AIM¹ and it could answer fact-based question by showing data found online (Carbone, 2016). Users could ask about sports results, the weather and stocks among other things (Carbone, 2016).

IBM Watson

IBM Watson was developed in 2006 specifically to compete with the champions of the game *Jeopardy!*. It won the competition in 2011 (Gabbatt, 2011). After that, IBM Watson became a public service for building chatbots for various domains which can process large amounts of data.

Smart Assistants

The smart assistant *Siri* was launched in 2010 by Apple. Other corporations followed the example: *Google Now* was made public in 2012, *Microsoft Cortana* in 2015, and *Amazon Alexa* also in 2015. All of these assistant can answer fact-based questions by performing an online search, set reminder and alarms, make calls, send text messages, etc. No implementation details are known about these applications, since they are closed-source and commercial.

Chatbots for Social Media Websites

Many businesses have utilised chatbots in their social media websites. They are used to automate tasks which generally require a person to contact the users. Such tasks include answering frequently asked question, accepting payments, booking appointments, etc. Chatbots are useful for businesses since they don't require payment and do not have set working hours but allow simultaneous communication with an unlimited number of users.

¹AIM stands for “AOL Instant Messenger”. It was a popular chat program in the USA in the early 2000s.

The results of study (Pereira & Díaz, 2018) on the 100 most popular chatbots on Facebook showed that such programs are developed for a specific purpose and do not try to imitate even basic human conversational skills.

Microsoft Tay

Microsoft Tay was launched in 2016 (Griffin, 2016) on Twitter as a chatbot the users of the social media website could send direct messages to or tweet. It was designed to mimic the language patterns of a 19-year-old American girl, and to learn from interacting with human users of Twitter and get progressively smarter (Price, 2016). However, Twitter users began tweeting Tay inappropriate messages and the chatbot began mimicking the behaviour and started using inappropriate language (Price, 2016). As a result, the bot was taken some of its tweets were deleted.

2.2 Artificial Intelligence

The first chatbots to exist utilised simple pattern-matching techniques to imitate intelligence. However, in recent years advances in the field of Artificial Intelligence have allowed developers to build more elaborate systems which try to replicate the ability of humans to learn. This chapter will introduce Artificial Intelligence and will describe its different branches. The techniques relevant to chatbots will be highlighted.

2.2.1 Definition

The Oxford dictionary defines *intelligence* as “the ability to acquire and apply knowledge and skills” (“Intelligence”, n.d.). Therefore, it can be expected that *Artificial Intelligence* (AI) would be the development of a machine which displays these qualities.

A more technical definition is suggested by Williams (1983): “a multidisciplinary field whose goal is to automate activities that presently require human intelligence”. Poole, Mackworth, and Goebel (1997) provide another definition: the study of the design of *intelligent agents* (or rational agents). Such agents receive percepts from the environment and perform actions based on them (Russell & Norvig, 2009).

Both definitions are similar as they talk about intelligence displayed by machines, however they refer to different applications of AI: intelligence and behaviour.

However, AI can be applied in many contexts. Williams (1983) summarises

them as:

- Perception - using sensors to analyse data from the physical world.
- Manipulation - interacting with the physical world.
- Reasoning - using human-like cognitive processes such as learning, understanding, etc.
- Communication - using natural language to exchange information with other entities.
- Learning - utilising experience to improve.

Lastly, (Poole et al., 1997) argues that AI aims to investigate intelligent behaviour in both natural and mechanical systems.

Several subfields of AI exist. The focus in this chapter will be on *Natural Language Processing (NLP)* and *Machine Learning* as they are relevant to the development of chatbots.

2.3 Machine Learning

Wojtusiak (2012) defines Machine Learning (ML) as “a scientific discipline that concerns developing learning capabilities in computer systems”. It combines several fields of science to achieve its goals: statistics, logic, robotics, computer science, computational intelligence, pattern recognition, data mining, cognitive science (Wojtusiak, 2012).

Traditionally, computer programs are given explicit instructions how to solve a specific problem. However, Machine Learning algorithms allow computers to learn from data. ML algorithms analyse the separate objects (instances) within the data with statistical methods in order to predict values (features) based on the training data.

An ML algorithm aims to define the correlation between an object x and a value y as a function f . This allows the algorithm to then predict what value would correspond to a newly introduced instance p . To define this correlation, an algorithm needs background knowledge, which it gains by observing examples of object-value mappings: the training data. Fig. 1 illustrates how a learning algorithm gains knowledge and creates a model of its input data. The models are utilised by an execution algorithm for prediction.

The different types of ML algorithms are based on the type of training data they require. They can be classified as:

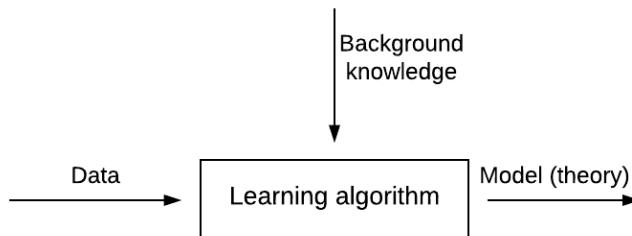


Figure 1: The process of training a learning algorithm. The algorithm analyses the data it receives as input, utilises previously generated knowledge (if it exists) and generates a model of the data. Adapted from (Kononenko & Kukar, 2007).

- Supervised learning - the program is provided with manually labelled training data from which it gains knowledge. “Label” refers to the expected output for each object.
- Unsupervised learning - unlabelled data is provided so that the program discovers how objects are mapped to values without human assistance.
- Semi-supervised learning - both labelled and unlabelled training data are used. The algorithm learns how the labelled data is classified and classifies the unlabelled data in the same way.
- Reinforcement learning - the system is rewarded when it generates the correct output, and is punished when the output is incorrect.

What follows is a description of each method.

2.3.1 Supervised Learning

Supervised Learning requires training data which is labelled. By utilising an algorithm, the system tries to understand the mapping between each item in the dataset and its label. If it successfully does so, when given more unlabelled data, it should be able to predict what label each item corresponds to. This process resembles human learning with a teacher.

The disadvantages of this method are that the preparation of the training data requires manual annotation of each item by a human. For large data sets this could be a time-consuming task. Additionally, if the data is specialised in some area, a professional might be required for correct annotation.

There exists several supervised learning algorithms. They can be categorised based on their goal: *classification* and *regression*.

Classification

Classification is a method where a system is taught how to map an object to an attribute. To learn how to do this, it is provided with training data in which objects are mapped to their corresponding categories (classes) correctly. By utilising a classification algorithm, the system can analyse the data and discover the logic behind the mappings. After the training process, the system can categorise new objects based on its knowledge and assign a class to them from its set of known attributes.

An example of classification would be using training data in the form of a list of RGB values of colours, where each value has the corresponding colour name assigned. After the system analyses them, it should be able to predict what colour new RGB values correspond to.

Some classification algorithms are:

- Naïve Bayes - based on probability. It assumes that the attributes of objects are independent within the context of the class (McCallum, Nigam, et al., 1998). This is the most basic classification algorithm, however it has been known to produce satisfactory results.
- K-nearest neighbour - in this method, the training data is stored. When an object has to be classified, it is compared to the known objects with a distance function and the closest matches' class is assigned to the new object (Kononenko & Kukar, 2007).
- Decision trees - the instances are sorted by their features in a tree-like data structure. The nodes of the tree correspond to attributes while the branches represent a possible value for the node. The objects are sorted by their attributes and the classification process starts from the root node (Kotsiantis, 2007).

Regression

In regression methods, the focus is prediction of numerical variables as opposed to the categories predicted in classification. These methods are usually used for the prediction of quantities, sizes, amounts, etc.

Some regression algorithms are:

- Simple linear regression - these algorithms aim to predict one numerical variable based on a single attribute. For example, predicting the price of a smartphone based on its year of manufacture.
 - Multiple regression - the goal is to predict one numerical variable based
-

on multiple features. An example is predicting the price of a smartphone based on its RAM, processor and storage capacity.

- Logistic regression -

2.3.2 Unsupervised Learning

Unsupervised learning methods do not use labelled data to gain knowledge. Instead, the goal is to discover how objects are related. The advantage of this method is that labelled data is difficult and time-consuming to produce and more unlabelled data exists.

In contrast to Supervised Learning, the aim is to find patterns within the data instead of prediction or classification (Kononenko & Kukar, 2007). More uncertainty exists within unsupervised methods, however, since the results are limited to the provided data. If the data is incomplete, meaningless correlations might be found between items.

Unsupervised learning methods can be categorised as:

- Clustering - aims to group the instances into coherent subsets (Kononenko & Kukar, 2007). Items within one cluster display more similarity than items in other clusters. The task of the algorithm is to define the grouping criteria and cluster the objects accordingly. Clustering is often used in statistical analysis of data. The K-means algorithm is a popular choice.
- Association rule-mining - used for finding relationships between items. For example, online shops utilise this method to suggest products to users based on what they have bought previously.

2.3.3 Semi-supervised Learning

Semi-supervised learning is useful in cases when a sufficient amount of labelled data is not present and it is difficult to acquire it. This method allows the usage of some annotated data and a larger amount of unlabelled data. Some possible techniques that can be used are:

- Graph-based methods
- Generative methods
- Heuristic-based methods

2.3.4 Reinforcement Learning

Reinforcement Learning is method that utilises a trial-and-error technique to learn in an interactive environment. This method be implemented in 3-dimensional simulation, etc.

The agent in the system is programmed with instructions on how to interact with the world. Based on its surroundings, it performs an action. If the action contributes to the agent reaching the state it is aiming for, it is rewarded. On the other hand, if the action it performs is incorrect, it is penalised. Based on the result, the agent then changes its behaviour in order to receive a reward. This process is repeated until the agent has generated a set of rules (knowledge) needed to achieve its goal.

The main difference with unsupervised learning is the use case. In reinforcement learning, the agent aims to define a set of beneficial actions needed to reach a pre-specified task while in unsupervised learning the system attempts to find patterns within its input data.

2.4 Natural Language Processing

Natural language processing (NLP) is an area where multiple seemingly unrelated sciences are combined: AI, linguistics and Computer Science. Goldberg and Hirst (2017, p. 17) describe Natural Language Processing as a “collective term referring to automatic computational processing of human languages”. This definition applies for applications which receive natural language as input and programs which produce human-like text as output. In addition, Goldberg and Hirst (2017, p. 17) highlight that such functionality is needed as communication with a computer using natural language would be much simpler for humans.

However, “understanding” language is a difficult task for machines, even though it is one of the first things humans learn. The problems occur due to the ambiguity of language and its constant evolution. Moreover, usually computers follow a set of instructions to perform a given task. However, language cannot be represented as a set of rules.

The initial approaches to NLP were simplistic and based on logical rules (e.g. ELIZA [Weizenbaum, 1966]). Modern projects employ *statistical methods* and *neural networks* (Goldberg, 2015).

For an NLP system to successfully operate, it has to analyse and preprocess its input. Martinez (2010) summarises the methods an NLP system might use to achieve this as:

- Morphological – concerned with the structure of individual words.
- Syntactic – refers to the structure of each sentence and the rules of constructing them.
- Semantic – refers to the meaning of a piece of text.

2.4.1 Morphological

Morphemes are the smallest part of a word which conveys meaning. For example, when analysing the word “unhappily” morphologically, it can be divided into the following morphemes: *-un* (prefix), *happy* (root) and *-ly* (suffix). Morphemes affect the meaning of words. For example, in the word “unhappily”, the prefix indicates that the opposite meaning of the root word is used, and the suffix shows that the word is an adverb. Another example would be words in the past tense which have the suffix *-ed*. Since prefixes and suffixes do not change, it is relatively easy for an NLP system to process them.

2.4.2 Syntactic

Syntax is concerned with whole words as opposed to morphemes. The syntactic analysis aims to define the grammatical structure of a sentence.

NLP systems utilise different methods of analysing a text syntactically. Some common ones are presented below.

Tokenisation

Tokenisation is a process which converts a document to a set of tokens and removes certain characters (Manning, Raghavan, & Schütze, 2008). Usually punctuation, possessives and apostrophes in shortened words are stripped from the text. Manning et al. (2008) also makes the point that each language has different rules for punctuation and word joining which makes the process of tokenisation very language specific.

Part of speech tagging

This process aims to assign tags (labels) to each word based on the part of speech it represents. Possible approaches to this are Markov chains or Hidden Markov Models (Martinez, 2010). The former is useful when there is a large amount of already tagged data present, while the latter is preferable when training data does not exist (Martinez, 2010).

Stop words removal

Stop words are words that occur too often in a text and convey little meaning (Martinez, 2010). An approach to identifying them is proposed by Manning et al. (2008): sorting all the terms by frequency of appearance and manually analysing the most common ones and storing the relevant entries (which convey no meaning) in a list of stop words. Every item in this list is then discarded when processing the text. A disadvantage of this method is the need for manual identification of the unnecessary phrases, although it could be more accurate than an automated one.

However, in some cases, stop words have an important role in a sentence, e.g. “Flight to London”. Removing the preposition “to” results in a loss of the meaning of the sentence (Manning et al., 2008).

a	an	and	are	as	at	be	by	for	from
has	he	in	is	it	its	of	on	that	the
to	was	were	will	with					

Figure 2: Example list of stopwords. Adapted from (Manning, Raghavan, & Schütze, 2008).

Stemming and lemmatisation

Both stemming and lemmatisation aim to fix the problem with different forms of a word. For instance, the words “studying” and “studied” should both be interpreted as a derivation of the word “study”, rather than separate words (Manning et al., 2008).

Stemming is the process of reducing a word to its root (Martinez, 2010) by removing its suffixes, affixes and prefixes. An example would be converting the word “studied” to its root by removing the suffix for past tense *-ed*. The result would be *studi*. However, stemming the word *studying* would produce *study* which will not match *studi*. The process, however, works well in some cases, e.g. *going* would be converted to *go* since the root does not change when the suffix “-ing” is added. Several stemming algorithms exist: Lovins stemming (Lovins, 1968), Paice stemming (Paice, 1990), and the most widely adopted algorithm in modern projects: Porter stemming (Porter, 1997).

Lemmatisation aims to be more precise than stemming. In this case, the word *studied* would be converted to the root *study*. However, to successfully do this, the program would need a comprehensive dictionary with all possible forms of a word.

Both processes have advantages and disadvantages. Stemming is easier to implement as it follows direct rules, however, depending on the use case, the results might not be satisfactory. Lemmatisation, while producing more accurate results, would be more difficult to implement and would require accurate data. Table 1 illustrates the output of both processes when they are given the same input.

Table 1: Comparison of the results of stemming and lemmatisation of some words.

Word	Stemming	Lemmatisation
dogs	dog	dog
am, are, is	am, ar, is	be, be, be
swam	swam	swim

Deciding which method to use would be based on the objectives of the software that is developed. For simpler cases, lemmatisation would perform well enough, however for more complicated use cases where accuracy is important, stemming would be the preferred approach.

2.4.3 Semantic

Semantics refer to the meaning conveyed by a sentence (Martinez, 2010). As described in the previous sections, both morphology and syntax contribute to the meaning. A common problem that has to be solved in NLP systems is *word-sense disambiguation*.

Word sense disambiguation

There exist words with multiple meanings. For instance, “right” could refer to a direction or it could mean that something is correct. Word-sense disambiguation aims to identify what meaning of such words was used in a sentence. Martinez (2010) presents the following methods to achieve this:

- Dictionary-based – lexical resources or thesauri are used. Generally, such methods
- Supervised – training data with manually pre-labelled words is used (the meaning of each word is the label). Popular techniques are Bayesian

classification and Information Theory. In terms of machine learning, this is a classification task.

- Unsupervised – clustering machine learning algorithms are used in this method. It is useful when no training data exists, or it is impossible to tag all the data. Some studies have shown it to perform at least as well as the supervised method (Yarowsky, 1995). - Dependency parsing

Only a subset of the techniques described above might be implemented, depending on the goal of the application that is being developed.

2.4.4 Approaches to NLP

Different approaches to implementing NLP exist. The most used ones are described below.

Rule-based

Rule-based methods are based on the belief that natural language can be represented as a formal model. It is arguable whether this is true as language was not created, rather it came to be. Furthermore, natural languages have numerous variations, constantly change and are difficult to define. The meaning of each word or combination of words cannot be defined as a set of rules (referred to as “grammars”). As an example, the rules can be expanded to include all meanings of the verb “drink”. However, rules for the nouns “drink” can be used in conjunction with must be added as well since “drink” can be combined only with nouns which refer to liquids. Creating such rules for every word in a language manually would be tedious, error-prone, difficult to manage and inefficient. For this reason, better techniques were required.

Rule-based methods were the first approach to NLP that was invented. An early chatbot which utilised this method was ELIZA. As Weizenbaum (1966) explains, ELIZA analyses its input and matches each word with a dictionary of keywords. If there is a match, a pre-made rule for this keyword is used to transform the sentence into appropriate output.

While rule-based methods have been shown to work, they do not

Statistical

Statistical approaches are based on calculating the probability of a string of words being a coherent sentence rather than comparing them to known words in a dictionary (Russell & Norvig, 2009).

Methods used to achieve this are:

- N – grams
- Naïve bayes
- Markov Chains
- Hidden Markov Models

2.5 Summary

This section introduced the roots of artificial intelligence with the ideas of Alan Turing and discussed the different branches of AI (Machine Learning and Natural Language Processing). The techniques which will most likely be used in the development of the application proposed in Section 1 will be a combination of Natural Language Processing and Machine Learning.

Since the chatbot to be developed will have to answer specific fact-based questions, supervised learning methods will be used. Therefore, factual data will be required for the training process. Such data exist in abundance, both pre-compiled sets of technical data and online forums where such data is stored.

3 Technology Review

This chapter will describe what technologies were selected for the development of the chatbot.

3.1 Chatbot Library

The first choice made was what library or service to use for input processing. Libraries which are made specifically for chatbot development were investigated. Unfortunately, few options exist. The results of the investigation were the following frameworks: BotMan, Chatterbot and several online tools for chatbot creation.

BotMan

BotMan is a framework written in PHP. However, it uses a pattern-matching method for input processing. A question and an answer can be specified. Whenever the chatbot receives a question has in its rules, it returns the response to it. Even though BotMan could be considered a viable option, its main use case is the creation of marketing chatbots. However, the aim of this project is to implement chatbot which can answer a wide variety of questions on different topics. Additionally, the pattern-matching method would be insufficient for the purposes of this project. Lastly, BotMan is advertised a marketing chatbot creation tool and this is what it is most suitable for.

Online tools for building chatbots

Many such services exist, e.g. Microsoft Bot Framework, Wit.ai, Dialogflow, IBM Watson, Pandorabots, Botpress. However, their use case is similar to BotMan's - the creation of marketing chatbots. Furthermore, such services are very limiting as their creation does not involve any programming. Lastly, these services are paid. While some do offer free plans, they provide extremely limited resources and limit the traffic to the application.

Chatterbot

Chatterbot is a Python library for creating general purpose chatbots. It utilises Machine Learning and NLP to match the user input to a database of known data in order to find the most similar statement. It is open source and extendible and has comprehensive documentation.

In conclusion, the Chatterbot library is the most suitable choice for the development of this project as it is extensible, utilises NLP to compare user

input to known data and is highly configurable.

3.2 Chatterbot

The main technology used for the development of the project will be the python module *Chatterbot*. It provides a simple way to generate a response to user input (Gunther Cox, 2019). It was specifically created in order to help programmers develop chatbots.

3.2.1 Overview of Input Processing

Fig. 3 illustrates how Chatterbot works on a high level. Firstly, the user inputs a statement (their question). It is then processed by the *logic adapters* of the chatbot and the response with the highest value of confidence is returned to the user.

3.2.2 Configuration

Chatterbot provides a `ChatBot` class. An instance of it takes several parameters in its constructor which define its behaviour:

- `preprocessors`
- `logic_adapters`
- `storage_adapters`

3.2.3 Preprocessors

The `ChatBot` instance can take a list of functions in its constructor to be used as preprocessors. Preprocessors can modify the input sent by the user before it is passed to the logic adapters. This is necessary due to the fact that the logic adapters are more likely to perform better if they receive clean input. For example, if the input contains a lot of white spaces or non-alphanumerical characters, it might be beneficial to remove them. Chatterbot has several preprocessors built in, however, custom functions can be created and used as preprocessors.

3.2.4 Logic Adapters

Logic adapters are used to select an appropriate response to a question entered by the user. More than one logic adapter may be used as different types of input may require different processing. For example, Chatterbot by

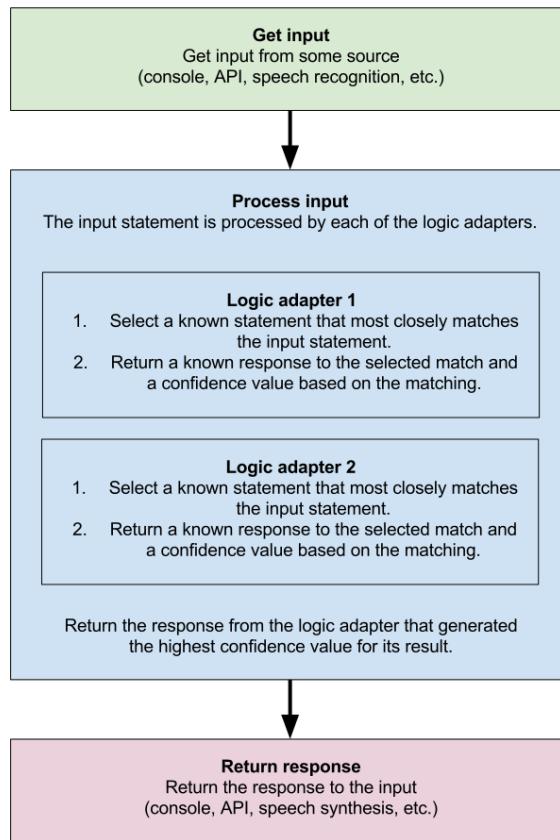


Figure 3: Diagram of the process of receiving input, processing it and generating a response (Gunther Cox, 2019).

default uses the `BestMatch` logic adapter. However, it is only useful when the question asked has a specific response. If the user asked for the current time, the `BestMatch` adapter would be unable to answer as there is no specific answer to this question, i.e. the answer changes depending on when the user asks the question. For this reason, a `Time` logic adapter can be created and added to the list of adapters used (Chatterbot has this adapter implemented by default).

To help decide which adapter to use if more than one are provided, a custom logic adapter must inherit from the base class `LogicAdapter` and must implement the method `can_process()`. This method is called before the logic adapters generate a response and the user input is passed to it. It must then return `True` or `False` based on whether the adapter can process this question. Listing 1 shows a possible implementation of the `can_process` method. This

example tells the adapter to only process user input which contains the string “Hey John” (John could be the name of the chatbot) in its beginning.

Listing 1: Example implementation of the `can_process()` method. Adapted from (Gunther Cox, 2019).

```

1 def can_process(self, statement):
2     # If the statements starts with
3     # the string 'Hey John',
4     if statement.text.startswith('Hey John')
5         # it can be processed with this adapter
6         return True
7     # If not,
8     else:
9         # it can't
10        return False

```

Another requirement is that custom logic adapters implement the `process()` method. It takes the input statement as a parameter. Additional response selection parameters can be added if required by the application. This method must return a `Statement` object. The `text` property of the object should contain the actual response to the input, while the `confidence` property should be a value between 0 and 1 which indicates how confident the adapter is that this response is correct.

In the event that more than one logic adapters are used, and more than one of them returns a response, the response with the highest value of confidence is selected. If two or more responses have the same value of confidence, then the response of the adapter which is first in the list of adapters is used. Fig. 4 show a diagram of response selection with multiple adapters.

By default, Chatterbot has the following adapters included:

- Best Match Adapter - used when question that have a specific response are asked. It requires training data and the quality of the response depends on the quality of the training data. The adapter uses a specific *similarity* function to search through the known questions. When it finds the most similar one, it then returns the known response to this question. The similarity functions available by default are:
 - ◊ Levenshtein Distance - this method is used by default. It calculates how many characters need to be changes for one string to become the same as another. It is used to compare 2 strings an it returns a value showing how similar they are.

- ◊ Jaccard Similarity
- ◊ Sentiment Comparison

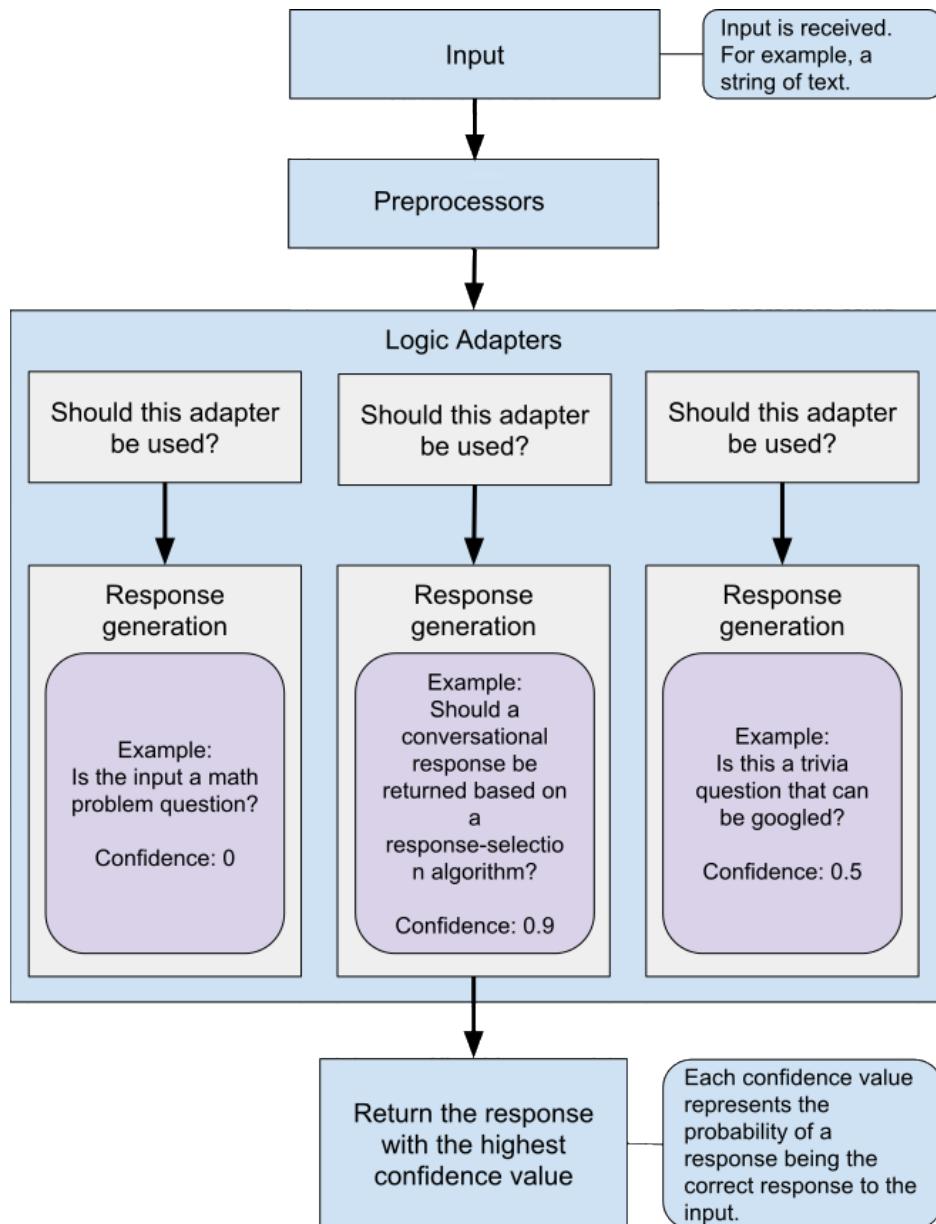


Figure 4: Process flow diagram of how responses are selected. In this example, the final response is the one generated by the adapter in the middle as its confidence is 0.9 which is greater than the responses from the other 2 adapters. Adapted from (Gunther Cox, 2019).

- ◊ Synset Distance
- ◊ Custom similarity functions may be implemented if needed.
- Time Logic Adapter - allows the user to ask about the current time.
- Mathematical Evaluation Adapter - calculates mathematical expression the user has entered.
- Specific Response Adapter - returns a specific predefined answer to a specific statement configured in the adapter.

3.2.5 Storage Adapters

Storage Adapters allow a ChatBot to use different storage technologies such as a database. The storage adapter used by default is the `SQLStorageAdapter`. It allows the chatbot to store its data in a local `SQLite` relational database. The structure of the database can be seen in Fig. 5. It consists of 3 tables:

- **Statement** - stores the processed training data. The field `text` contains each response as it is seen in the training data. `search_text` stores the processed response text, i.e. stop words are removed, a basic form of word-sense disambiguation is applied where each word is replaced by its root or a synonym and each word is labelled with its corresponding part of speech. `in_response_to` contains the question that each response was assigned to in the training data. Lastly, `search_in_response_to` contains the processed version of each question from the training data.
- **Tag** - each unique category found in the training data is added to this table.
- **Tag association** - shows what category each statement is assigned to.

3.2.6 Training

Training a ChatBot instance is achieved with the help of training data which represents conversations. The data is analysed, processed and each statement and its corresponding answer are added to the database of the chatbot. The processing involves removing stop words, removing punctuation, tagging each word with its corresponding part of speech and replacement of some words with synonyms.

Chatterbot includes the following trainers:

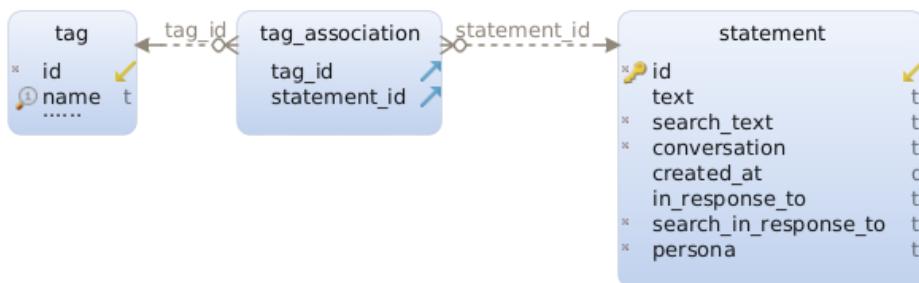


Figure 5: The table structure of the relational database used to store the “knowledge” of a chatbot.

- List Trainer - trains the chatbot with data stored in a list.
- Corpus Trainer - trains the chatbot with data stored in an external file.

Custom trainers can be implemented. They must extend the class *Trainer* and implement the method `train()`.

3.2.7 Generating a response

After the ChatBot instance has been initialised and trained, a response can be generated by calling the method `get_bot_response()`. It takes a string containing a question as a parameter and returns the response of the chatbot.

The response generation is achieved by preprocessing the question with the same NLP techniques used when training data is learned. A response can then be retrieved from the database by using a SQL SELECT statement which finds a statement with a value for `in_response_to` which matches the processed input.

3.3 Python

Since the main library used for the development of the project is written in Python, the decision to implement the application in the language was made.

Python is known to greatly reduce development time as it provides a lot of “syntactic sugar”. Another advantage is its automatic garbage collection. In traditional programming languages such as C++ manual memory management is an issue for new programmers and security and performance issues might occur when it is not implemented correctly.

Python is a weakly-typed language, which means that variable types are deduced by the python interpreter. For this reason, developers do not have to specify the type of variable they are declaring. However, this could lead to problems as the type of data stored in each variable has to be remembered since assigning a different type later would result in an error. Moreover, Python is a high-level language, meaning that it provides more layers of abstraction. This results in the writing of less code to solve a specific problem.

Python has a large selection of modules. Additionally, Python is a multi paradigm language, i.e. it supports object-oriented programming, functional programming, imperative programming and procedural programming. Lastly, it is a cross platform language, being able to run on most major operating systems.

Similarly to other programming languages, Python does have disadvantages. The main one is that it is an *interpreted* language. This means that rather than compiling the code to an executable file (machine code), the python interpreter reads it line by line and executes each instructions. Another problem caused by this is that errors can only be found during the runtime of a program. If a function of the software is rarely executed, issues with it might not be discovered. This issue can be solved relatively easily. For example, unit tests can be written which execute all functions of a program. Even though this problem doesn't exist with languages such as C++, where errors are reported during compilation, programs still have to be tested for logical errors.

3.4 Flask

Another important decision that had to be made was what type of application to develop. The options considered were: a smartphone app, a program which targets desktop operating systems (Windows, Linux or Mac) or a web app. Since the main goal was to make the deliverable accessible from any platform, the idea of smartphone app was disregarded. Furthermore, the main mobile platforms have very different development tools and processes. A desktop application was not implemented for the same reasons - it would only work on one type of device and targeting all desktop OSs would be a difficult task. In contrast, a website, would be accessible from any device which can be connected to the Internet and has a browser program. Incidentally, all the devices mentioned above, mobile devices and PCs, have this functionality.

The technologies considered initially were PHP for the development of the website and MySQL for database interaction. However, since the goal was to implement a simplistic website, it was decided that the difficulty of setting up a PHP server would outweigh the benefits. It was also unclear how the communication between the Python library and a PHP program would be implemented. Instead, methods for web development in Python were investigated. The results showed that Python has become a popular choice for website creation with its 2 popular frameworks Flask and Django. Ultimately, the decision to use Flask was made as it provided a smaller subset of tools which was enough for the creation of a simplistic web app. Furthermore, Flask was easier to set up and configure.

Flask is a micro framework for Python used for website development (Flask, 2019). It is described as a microframework as it aims to only include the core functionality required to create a website. For instance, by default, Flask does not include any database interaction functionality. However, it would not be required since it is implemented by Chatterbot.

Flask includes a server which can be used for development, page routing and templating.

The page routing functionality allows Flask to create multiple pages within one website and users to access them.

The templating engine extends the functionality of HTML. Templates are HTML files which can include special tags which are processed by the Flask server. One common usage is passing variables from a Python script to the HTML code so that its value can be displayed. When the user accesses a page, its template is converted to static HTML by the template engine (Jinja) and it is displayed to the user. Templates also allow the reusability of code since one template can extend another, i.e. the contents of the former can be automatically appended to the templates of the latter.

3.4.1 Twitter Bootstrap

Since the goal was to implement a website which could be used from any device which can access websites, Twitter Bootstrap was used. It is a CSS framework which provides a responsive design functionality. It allows the positioning elements in a grid of rows and columns. When the page is resized, the columns are re-ordered and their size is changed dynamically to fit the page. The font size is also modified based on the resolution of the device.

3.5 Pytest

As explained above, one of the disadvantages of Python is that errors might not be discovered until the code is thoroughly tested. However, often it is not possible to test all aspects of a program manually after making changes to the code. For this reason, an automated testing system was planned. Unit tests were a method to achieve this. The Python library Pytest was selected as the unit test framework since it is recommended by the Flask documentation (Flask, 2019) and it is supported by Flask.

3.6 Web Hosting

Since the project implements a website, web hosting options were considered so that the website can be made public. Traditional web hosting options could not be used as they offer PHP servers with MySQL databases. Instead, web hosting services which support Python and Flask were considered. The main options were:

- Amazon AWS
- Heroku
- Python Anywhere

All these services are viable options, however both AWS and Heroku have a limit of requests for their free tier. If the limit is exceeded, payment is taken from the website owner's account. Python Anywhere, on the other hand, is free. Furthermore, it is the easiest to set up as it supports Flask websites by default.

4 Implementation

During the Interim Meeting in week 9 of semester 1, the second marker of this project proposed that deliverable could have a specific purpose to simplify the evaluation process. Evidence of this is provided in Appendix D (see “Any other recommendations as to the future of the project?”). The advice was taken and the objectives of the project were changed. The initial idea can be seen in Appendix A. Section 1.2 (Aim and Objectives) introduces the new project aims. Therefore, this section will discuss the implementation process of the new project.

4.1 DevOps

This section explains the development operation methodologies used during the implementation process.

4.1.1 Management

An Agile workflow was adopted. Each week was considered a sprint and a working build of the application was deployed at the end so that it can be demonstrated during the meetings with the supervisor of the project. A prioritised sprint backlog was created for each sprint with the application Trello. An example backlog can be seen in Fig. 6.

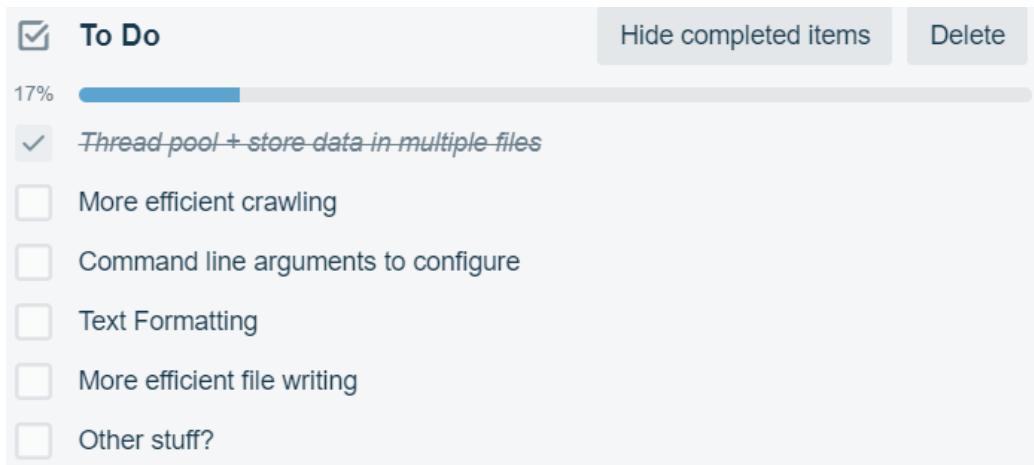


Figure 6: An example of a prioritised sprint backlog. The items on top have a higher priority and should therefore be implemented first.

Additionally, a project plan was created in the form of a Gantt Chart (Appendix B). Goals were set for each iteration and milestones were defined.

4.1.2 Version control

Version Control was used in the development of the project. The version control system used was *Git* and the repository was hosted on *Github*. The reason for this was previous experience with these technologies.

Using Git and Github meant that the source code of the application could be easily shared between computers. Additionally, changes between versions of the program could be tracked. In the event that issues arose, the application could be reverted to a previous working version. If necessary, the difference in the source code between two versions could be inspected.

Another feature of Git and Github used was *branches*. Branches allow for having several versions of the code at the same time. The following branches were created for this project:

- Master branch - a stable version of the application. It is thoroughly tested and no known bugs are present. The purpose of this branch is to contain the version of the code which can be deployed.
- Development branch - the most recent updates to the application which might not be fit for deployment. It can be updated until it is considered stable enough. It can then be merged into the master branch and deployed.
- Feature branches - a branch where a specific feature is implemented without affecting any of the other versions. It can be used for testing ideas or implementing a major new feature.

Using branching proved to be beneficial as the project was made publicly available early in the development process. Since continuous deployment was implemented, having a single branch was insufficient as unstable changes would cause problems in the live version of the website.

4.1.3 Testing

Since the application consisted of many components communicating with each other, problems were often encountered during the development process. For example, the chatbot expects to find the training data in a specific location. Therefore, the web crawler has to store the data in this exact location. To ensure changes would not break the current functionality, unit tests were implemented with the Python library PyTest.

The first set of unit tests written was for testing the command line interface. Many modifications and updates were made to the web crawler during the

development process. This always resulted in problems with the command line interface functionality as most of the commands modify the output of the crawler.

The second set of tests was for the chatbot functionality. These tests use PyTest fixtures which set up the necessary conditions before the tests are executed and delete any files or data left after they are run. These tests ensure the output produced by the chatbot is correct.

The last set of unit tests implemented was for the storage adapter. Its ability to interact with the database is tested.

4.1.4 Code Coverage

The PyTest library supports extensions. One such extension is `pytest-cov` which generates code coverage reports. Code coverage is a metric used to specify how many lines of code are executed when unit tests are run. For example, a code coverage of 100% means that all of the code in the project is tested. Therefore, the higher the code coverage, the better.

The `pytest-cov` coverage reports show detailed information about each file in the program that is tested. The code coverage for each file is shown as well as how many lines of code were not tested. At the end of the report, the average code coverage is shown. An example report can be seen in Listing 2.

Listing 2: A code coverage report produced with `pytest-cov`.

Name	Stmts	Miss	Cover
<code>chatbot/_init_.py</code>	49	24	51%
<code>chatbot/bot.py</code>	86	9	90%
<code>chatbot/config.py</code>	3	0	100%
<code>chatbot/constants.py</code>	22	0	100%
<code>chatbot/crawler.py</code>	80	6	92%
<code>chatbot/logic.py</code>	69	13	81%
<code>chatbot/models.py</code>	29	1	97%
<code>chatbot/run_it.py</code>	2	0	100%
<code>chatbot/storage.py</code>	199	51	74%
TOTAL	539	104	81%

The aim for this project was to achieve a code coverage of at least 85% which was accomplished. Since a lot of libraries were used, it was impossible and infeasible to test all the code in the project. It was assumed the creators of

libraries have tested their code thoroughly. Moreover, most libraries provide their own unit tests.

4.1.5 Continuous Integration & Deployment

A Continuous Integration service could be used to automatically execute the unit tests every time new code is added to the GitHub repository. Changes were made to the project daily, however running the tests after each change was time-consuming. Using Continuous Integration meant that every time a commit was pushed to the GitHub repository, the tests would be run automatically, and any issues would be found quickly. Furthermore, since branches were used, the tests could be run after merging changes into the master branch. This allowed for the discovery of integration problems.

The Continuous Integration service used was Travis CI due to previous experience. It is used for continuous testing and delivery of projects hosted on GitHub. To set it up, the GitHub account which contains the project repository was linked to Travis CI.

Configuration of the build process is achieved by adding a “.travis.yml” configuration file in the root directory of the repository. The file is written in the YAML format. Firstly, the operating system used to compile the project is specified. In this case **Ubuntu 16.04 (Xenial Xerus)** OS is used. Additionally, the language of the program was developed in is specified, i.e. Python. The versions of the language can be specified next. In this case, the project is tested with Python 3.6 and 3.7 which are the latest versions at the time of writing. Travis CI also supports the following steps:

- **install** – this step is executed before the testing happens. It is used to install any packages that are not installed on Ubuntu by default. In this case, there are no packages that should be installed, however the libraries used in the project are downloaded via the package manager **pip** package manager.
- **before_script** – this step occurs before the test script is executed. It is used to set the environment variables Flask expects.
- **script** – this step executes the command which runs the unit tests. In this case, it is the **pytest** command. No other setup is required as pytest is able to discover unit tests without any additional configuration. However, since code coverage reports can be generated, the optional parameter **--cov** is added so that a coverage report is created.
- **after_success** – this step is only executed after the tests have com-

pleted without any errors. The coverage report generated in the previous step is uploaded to codecov.io. It can then be inspected in detail.Codecov.io provides in-depth reports and charts of code coverage. Each file in the repository can be seen and it is shown whether each line is tested. This is useful for finding possible problematic areas which are not tested.

- **cache** – this option specifies dependencies or folders which do not change often. They can be cached, instead of being downloaded every time the build process is run. In this case, the `pip` dependencies can be cached which speeds up the testing process.

Continuous Deployment

Since the website was hosted on Python Anywhere, Continuous Deployment could be implemented. This was done for faster deployment every time code was added to the master branch. This had to be implemented manually since neither Python Anywhere nor Travis CI had any methods of deploying Flask websites.

The first step was to create an end point in the website. It was created at “/webhook”. Its purpose was to receive requests from GitHub. Additionally, GitHub’s webhook feature was configured to send a request to the live website’s “/webhook” address whenever a commit was added to the master branch.

The website is configured to inspect the request body when it receives a request. It is checked whether the event that occurred is a commit and whether the commit was made to the master branch. If this is the case, changes from the repository can be pulled directly. This was achieved with the Python library GitPython. Unfortunately, when changes are made to the website, Python Anywhere requires the site owner to reload the website. This has to be done manually by going to the site settings and pressing a “reload” button. Python Anywhere has an API which provides an end point for reloading a website remotely, however, this cannot be done from the same program that runs the server as the request times out due to the server going offline for reloading.

4.2 Application

One of the main early considerations was how to structure the source code. It became clear that the application should consist of two main parts: a back-end and a front-end program. Since the user will interact with a user

interface (the front-end) and another program will process their input, it was logical to separate the two.

4.3 Back-end

The back-end program consists of a web crawler and a response generation program. The former is used to collect training data from Stack Overflow, while the latter processes the user input and generate a suitable response.

4.3.1 Web Crawler

Before creating the back-end program, it was necessary to collect training data for the chatbot. The possible sources of data were online resources and books.

There exist numerous books about programming languages and techniques. However, it was decided that this approach was infeasible as the Chatterbot library required training data in the format *question - answer*. Extracting data from a book into this format would require advanced knowledge of NLP and the development of another program for this specific task.

Another option for data collection was websites which provide beginner tutorials about programming. However, limited amount of information of questionable quality was mainly available.

Ultimately, the decision to scrape¹ the website Stack Overflow was made. Stack Overflow is one of the most popular websites where programmers can ask questions about any programming language. Furthermore, there is a voting system which sorts the answers by number of votes. Additionally, the person who asks a question is able to mark a specific answer as the “accepted answer”. This meant that when collecting data, the answers to each question could be sorted by number of votes, and the “accepted answer” could be used as the default response of the chatbot. Unfortunately, some questions have no accepted answer or no answers at all.

It was decided that data about the C++ programming language would be collected for the prototype of the application. It is a language often used at universities to introduce students to advanced concepts of computing. However, new programmers often find it difficult to understand concepts such as pointers, references, memory management.

For the data collection process, a program was written with the Python module *BeautifulSoup*. It needs the following parameters to operate:

¹Web scraping - a technique for extracting information from a website.

- `start_page` - the first page to be crawled.
- `num_pages` - total number of pages to be crawled.

With these values, additional required parameters can be calculated: the last page to be crawled is the sum of `start_page` and `num_pages` and current page (indicating which page is currently processed) is initially set to the value of `start_page`. With this data, a loop can be executed, starting from the first page and running until the last page. In each loop iteration, a URL is generated to the corresponding Stack Overflow page. The structure of a Stack Overflow URL can be seen in Fig. 7. The output of the page can be modified by changing the values of the highlighted parameters:

- **base URL** is a constant value.
- **tag** is used to filter questions based on their category. In this case, C++ questions are requested as “%2b” expands to the symbol “+”. However, the tag can be changed to any language. For instance, to collect data about Java, the tag parameter can be given the value “java”.
- **sort** configures the order in which the questions are shown. In this case sorting by **votes** results in the most upvoted questions being shown first.
- **page** indicates the currently displayed page of results. Since there are numerous questions on the website, it is impossible to display them in one page.
- **pagesize** indicates how many questions are shown per page.

To generate a URL, all the parameters are appended to the base URL, and the **page** parameter is dynamically generated based on the `current_page` variable.



Figure 7: The structure of a Stack Overflow URL. Variables and their values are appended to the base url. The separator used is &.

With the generated URL, the page can be accessed from within a Python program with the library `requests`. The returned result is the source code of the page at the given URL. Data can then be extracted from it using the

Beautiful Soup Python library. Firstly, an instance of the `BeautifulSoup` class is created. The source code retrieved earlier is passed to it. The library can then parse it and create a *parse tree* which as an efficient data structure for storing hierarchical data.

The data needed from each page of questions is the hyperlink to each question. To extract this data, the method `find_all()` of the `BeautifulSoup` object can be used. It returns a list of all the occurrences of a specified HTML tag. In this case, the HTML `<a>` tag is needed as it is used to create hyperlinks. However, there are numerous hyperlinks on each Stack Overflow page. Only the hyperlinks which lead to a question page are needed. Using the Chrome browser developer tools, it was discovered that each link to a question was assigned a class value “question-hyperlink”. Listing 3 shows the HTML code of an example hyperlink to a question.

Listing 3: An example hyperlink to a Stack Overflow question.

```
1 <a href="/questions/121162/what-does-the-explicit-
    keyword-mean" class="question-hyperlink">What does
    the explicit keyword mean?</a>
```

With this information, only the URLs of the questions could be extracted. Listing 5 shows the source code for extracting the hyperlinks to each question. The `find_all()` method on line 3 returns a list of all `<a>` tags with a class value “question-hyperlink”. A for-each loop iterates through this list. Each hyperlink’s destination is extracted from the `href` parameter using the `get()` method. Each link’s text is also extracted using the `get_text()` method. The destination URL, however, is incomplete and it needs to be appended to the base Stack Overflow URL. The final question URL and the question title are passed to the `parse_question()` method. The if statement on line 4 is needed as there is a “Hot Network Questions” section on each Stack Overflow page which contains hyperlinks to popular questions. The if statement ensures these links are not processed.

The `parse_question()` method can be seen in Listing 4.

Listing 4: Extracting answers for a specific Stack Overflow question.

```
1 def parse_question(url, title, data):
2 # page to be scraped
3 page = requests.get(url, headers=headers, timeout=(3,
4 30))
5 soup = BeautifulSoup(page.content, 'lxml')
6 # get the question data, contained in a <div> with class
7 "postcell"
8 question = soup.find('div', class_='postcell')
9 if question is not None:
10 answers = soup.findall('div', class_='answercell')
11 # limit to max 3 answers per question
12 end = len(answers)
13 if end > CRAWLER.NUMANSWERS:
14 end = CRAWLER.NUMANSWERS
15 # for each answer found
16 for i in range(0, end):
17 # get the answer text
18 answer = answers[i].find('div', class_='post-text').
19 extract()
20 # store the question and the answer in their own list
21 answer = str(answer)
22 entry = [title, answer]
23 # add to the main list
24 data.append(entry)
```

Listing 5: Extracting the URL of each question on a Stack Overflow page.

```

1 # get a link to each question
2 for ques_link in soup.findall('a', {'class': 'question-
    hyperlink'}):
3 # make sure no extra links are crawled
4 if q_no == PAGE_SIZE:
5     break
6 # generate the link
7 url = SO_URL + ques_link.get('href')
8 # print question title for debugging purposes
9 title = ques_link.get_text()
10 # parse this question
11 parse_question(url, title, data)
12 # keep track of current question number
13 q_no += 1

```

The *requests* library is used again to send a GET request to each of the question pages. The returned source code is then passed to an instance of *BeautifulSoup*. Using the Chrome browser’s development tools it was discovered that the data about a question is stored in a HTML *<div>* with a class value “postcell”. However this includes all data including the date the question was asked, the author, etc. The text of the post is stored in a *div* with class value “post-text”. Each answer is stored in a *<div>* with class value “answercell”. The actual text of the answer is stored in a *<div>* with class value “post-text”. With this information, the answers to each question can be extracted by first finding them with the *find_all()* method. The resulting list is then iterated through and each answer’s text is extracted. The maximum number of answers scraped is currently set to 3. However, the if statement on line 12 ensures that in the event that less than three answers exist, no errors occur. Lastly, the question title and the top three answers are stored in a Python *dict*.

The last step is to save the data stored in the dict to a file. The ChatterBot library supports training from external files by default. Custom trainers can be created to support different file formats, however the default *YAML* format was used. The structure of a training file can be seen in Fig. 8. The “– –” tag is used to signify a statement, while the “–” tag signifies an answer to a statement. To save the data collected from Stack Overflow in this format, the *Ruamel.YAML* Python library was used. To ensure the data was saved in this required format, however, it had to first be stored in an appropriate data structure. Each question - answer pair was stored in a Python list. Each

list was appended to a Python dict. Using the Ruamel.YAML library's `yaml` class, the data could be written to a file using the `yaml.dump()` method. The code used to achieve this can be seen in Listing 6.

```

1 categories:
2   - trivia
3     conversations:
4       - - Who was the 37th President of the United States?
5         - Richard Nixon
6       - - What year was President John F. Kennedy assassinated?
7         - '1963'
8       - - The Space Race was a 20th-century competition between what two Cold War rivals,
9         | for supremacy in spaceflight capability?
10        - The Soviet Union and the United States.
11        - What was the name of the first artificial Earth satellite?
12          - Sputnik 1

```

Figure 8: Example training file in the YAML format.

Listing 6: Writing scraped data to a file in the YAML format.

```

1 # open out file
2 with open(out_file, 'w+', encoding="utf-8") as outfile:
3     # write data
4     yaml.dump(final_data, outfile)

```

The resulting program worked well, however it was slow as only one question was processed at a time. To improve its performance, multithreading was implemented. This was achieved with a `ThreadPoolExecutor`. It allows the creation of asynchronous tasks. Listing 7 summarises the process. A `ThreadPoolExecutor` is created and its maximum number of threads is set. Threads are created and started and they are instructed to execute the `crawl_pages` function. For each thread a suitable starting and ending page is calculated so that the work is split evenly. For example, if 10 pages were to be crawled with 2 threads, each thread will be assigned 5 pages. Additionally, thread 1 will start at page 1 and stop at page 5, while thread 2 will start at page 6 and stop at page 10.

Listing 7: Multithreaded optimisation of the web crawler.

```

1 with ThreadPoolExecutor(max_workers=workers) as executor
2     :
3         for i in range(workers):
4             executor.submit(func, (i * num_pages + 1))

```

Fig. 9 summarises how one thread of the web crawler operates. When multiple threads are used, the same process occurs for each thread simultaneously.

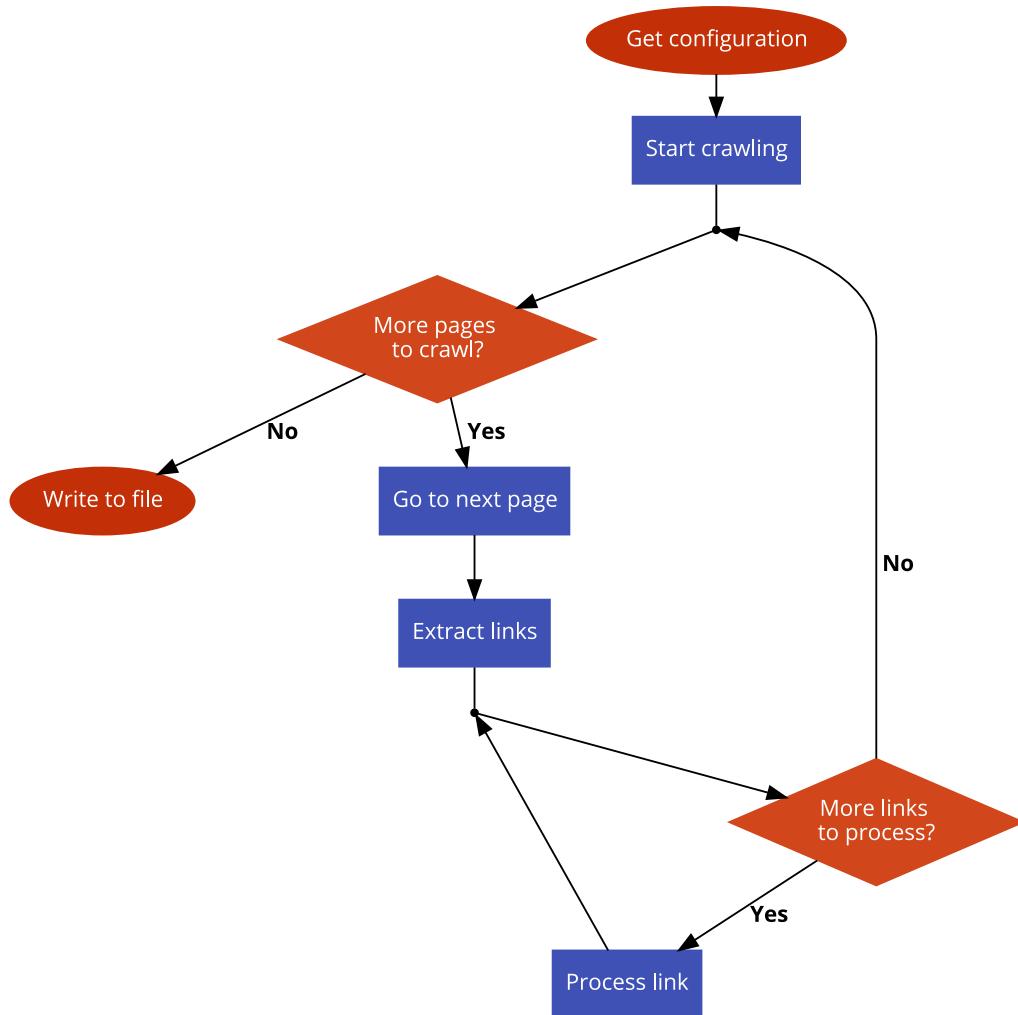


Figure 9: A flow diagram of the data collection process.

4.3.2 Chatbot

The other main part of the back-end program is the script which processes user input to generate a response.

An instance of ChatterBot's `ChatBot` class was created with the following configuration parameters:

- `preprocessors` - a list of the preprocessors the chatbot should use. The `unescape_html` preprocessor was used since it converts escaped HTML characters their symbol equivalents. For instance, "" is converted to the `` tag. This preprocessor was needed as the data

collected from Stack Overflow includes HTML tags and some of them are escaped.

- `logic_adapters` - the Best Match and Specific Response logic adapters were used.
- `database_uri` - an SQLite database was used.

The ChatBot object was then trained with the previously collected data. At this stage, it could only interact with a user through a terminal.

4.4 Front-end

The front-end was implemented in Python with the Flask module.

4.4.1 Project Structure

As suggested in the Flask documentation (Flask, 2019), the project was structured with modularity in mind. The structure can be seen in Listing 8. The following directories were created:

- `Chatbot/` - the root directory of the project. It contains all subdirectories and configuration files.
- `chatbot/` - contains the directory in which the Python code for the back-end program is stored.
- `static/` - contains the static files used by the front-end, i.e. CSS styles, JavaScript code, images and icons.
- `tests/` - contains the unit tests for the application.
- `venv/` - contains the Python *virtual environment* used for the project.

Structuring the project in this way ensured that scripts were separated based on their goal and it was easier to maintain the project.

Listing 8: The file structure of the project.

```
Chatbot/
|---requirements.txt
|
|---chatbot/
|   |---__init__.py
|   |---constants.py
|   |---crawler.py
|   |---bot.py
|
|   |---static/
|       |---css/
|           |---style.css
|
|       |---js/
|           |---chatbot.js
|
|---templates/
|     |---base.html
|     |---chatbot.html
|
|---tests/
|---venv/
```

4.4.2 Application Setup

The Flask documentation recommends organising the code in modules. For this reason, the `__init__.py` file is placed in the `chatbot/` directory (Listing 8). The existence of this file Python interpreter that `chatbot` is a module which can be imported in Python files. Additionally, this script defines a method which initialises an instance of the `Flask` class. This method is known as the *Application Factory* (Flask, 2019). The initialisation of the Flask app is achieved as follows:

1. An instance of the `Flask` class is created.
 2. The instance is configured. A `SECRET_KEY` variable is required by `Flask`. It is used for encryption of sensitive data such as passwords. During development, its value is set to the string “`dev`” so that debuggers can read encrypted information.
 3. The pages of the `Flask` application are defined. The code for creating
-

the home page can be seen in Listing 9. On line 1, the address of the page is specified. The “/” indicates that this is the home page and is shown by default when the website is accessed. If a Contact page were to be added, its location would be “/contact” and it would be accessed by navigating to the address `website.com/contact`. Line 2 defines a Python function which will be called when this address is accessed. Line 3 uses the Flask method `render_template()`. It requires the name of an HTML file which will be shown when this page is accessed. The HTML file is stored in the `templates/` directory. This method takes any number of other optional arguments. Every other variable passed to it will be forwarded to the template specified and its value can be used via the Jinja templating language. In this case, a ‘title’ variable is passed which specifies the title that will be shown in the page.

4. The last requirement is that the Application Factory returns the final Flask object.
5. The website can be run using the default server that comes with Flask. To do this, the command `flask run` can be executed from a terminal. Flask uses environment variables to locate a project. The variable `FLASK_APP` must be set before the website can be accessed. To run the development server in debug mode, the variable `FLASK_ENV` may be set to “development”.

Listing 9: Initialisation of the home page of the front-end application.

```
1 @app.route("/")
2 def home():
3     return render_template("chatbot.html", title="Chatbot")
```

4.4.3 Server-side Template Rendering

The main goal for the website was to create a clean and simple interface which is easy to use and feels familiar. Messaging apps such as Facebook Messenger and WhatsApp were used as reference. The initial website interface is shown in Fig. 10. It was created with Jinja templates. When the user opens the website, the Flask server converts the templates to valid HTML which can be displayed by web browsers.

A main template (`base.html`) defines the structure of the webpage and implements the elements which exist on every page of the website (e.g. headers, footers, menus). In its `<head>` tag all the CSS styles and JavaScript scripts are loaded. Its `<body>` tag only contains an HTML `<h1>` element which has

the value of the variable `title` passed from the Python code. Listing 10 shows how this was achieved. Line 1 checks if the variable `title` has a value. If it does, it is used on line 2 as a heading. There is no other content in the `<body>` tag, except for a Jinja `block`. It defines a place in the page where content can be inserted. This can be achieved by creating another template which implements the code which should be in place of the empty `block`.



Figure 10: The interface of the website.

Listing 10: Using a variable passed from Python in a Jinja template.

```
1 {% if title %}  
2     <h1>{{ title }}</h1>  
3 {% endif %}
```

Aanother template, `chatbot.html`, implements the elements specific to the chatbot page: the chatbox, the text input field and the “Submit” button. It extends the `base.html` template. As a result, the elements defined in the base template are shown when the chatbot template is rendered and any additional elements can be inserted by defining a Jinja `block`.

Styling

The styling of the website was achieved with CSS. The CSS framework *Bootstrap*

strap 4 by Twitter was used to style the website and to ensure that it can be viewed on devices with different screen sizes with no issues.

Although Bootstrap does not require any specific setup or configuration, the Flask extension Bootstrap-Flask can be used to implement a bootstrap layout more easily. It creates Jinja macros such as `bootstrap.load_css()`, `bootstrap.load_js()` which automatically load the required stylesheets and JavaScript which Bootstrap uses.

The chatbot page consists of three rows with one column per row. The title is positioned at the top of the page and the text input box and the button are positioned at the bottom. The chatbox is in the center of the page and it occupies all the remaining free space. By default, Bootstrap uses rows with the same height. However, custom css was written to position the elements more precisely and to change their height. To ensure the responsive design still worked, appropriate CSS size units such as “%” and “vh” (viewport height) were used. The reason for this was that specific sizes, e.g. “20 pixels” are constant regardless of the screen size.

4.4.4 Client-side Template Rendering

Most instant messaging websites do not refresh their webpage when a message is sent or received. However, when implementing the input functionality, it was discovered, Flask does not provide a mechanism for dynamic manipulation of a website’s content. Instead, templates are rendered to HTML once when the user accesses a page. To achieve dynamic message sending, JavaScript had to be used.

JavaScript is a client-side programming language which runs inside web browsers. It can be used to dynamically change a website’s content, store information and handle requests and responses. The JQuery library was used as it provides wrappers around JavaScript functions which allow for writing of less code to achieve the same functionality. It is already included in the HTML file as it is required by Bootstrap, so it could be taken advantage of its functionality.

This cannot be achieved with Flask since the page would have to be reloaded every time a message has been sent. Flask does not provide a mechanism for dynamically changing a website. However, this would introduce several problems such as loss of data when the page is reloaded and worse overall user experience. Additionally, the website would feel less responsive and slower. For these reasons the, JQuery DOM manipulation functionality was used.

Whenever a message is added to the chatbox (a user or a bot message), a piece of HTML code is added to the current content of the chatbox. Listing 11 shows the HTML code for creating a message by the bot. On line 1, a div is defined. Its class value tells Bootstrap how to style the div: the background colour is set to dark grey, the text colour is set to white and rounded corners are used for the background. The code for creating a user message is the same, except for the background colour. Every time the user inputs text or the bot answers an input, this code is added to the HTML of the chatbox.

Listing 11: The HTML code used to create a chatbox message sent by the chatbot.

```
1 <div class="p-3 mb-2 bg-dark text-white rounded">
2 I 'm fine , thank you .
3 </div>
```

The first attempt of implementing this functionality used constant JavaScript strings which contained the necessary HTML code. The changing content, the message being added to the chatbox and the background colour, could be changed by appending several variables to the contents of the string. However, this turned out to be confusing, difficult to debug and maintain. It was soon realised that it could be implemented using templates. Jinja templates seemed to be suitable for the task, however it was realised that Jinja could not be used in this case. Since the templates are rendered in the back-end when the user accesses the page, the templates could not be modified dynamically once the page is loaded. The reason for this is that Jinja is running on the server (the back-end), while JavaScript runs on the client (the user's browser).

After researching the possible solutions to the problem, it was discovered that there exist templating engines for JavaScript. They provide the same functionality as Jinja, however they work on the client-side after the page is rendered. One such library is JsRender. It was ultimately selected due to its simplicity and speed of rendering templates as well as its syntax being very similar to Jinja.

The code which would be reused had to be defined in one of the existing HTML files. It would have been preferable to store the templates in their own files as it would have provided better structure and separation of code. However, it was not possible as JsRender only supports such functionality on node.js servers. Instead, the templates were stored in the `chatbot.html` file. A requirement of JsRender is that templates are stored in an HTML `<script>` tag. The code for the message template can be seen in Listing 12.

ing 12.

Listing 12: The code for the JsRender message template.

```

1 <script id="messageTmpl" type="text/x-jsrender">
2 <div class="p-3 mb-2 bg-<%:bg_col%> text-white rounded">
3 <%:text%>
4 </div>
5 </script>
```

On line 1, the template is defined. It can be referenced by the “id” value it is set. The value for `type`, `text/x-jsrender`, helps the library identify templates. Since this is not a standard script type (e.g. `text/javascript`), it is not rendered by browsers as they ignore types they do not know by default. On line 2, the HTML `<div>` with required class values is defined. Where normally the background colour of the div would be set, a JsRender variable is used. A colon before the name of the variable is used to signify that it is a JsRender variable. Its value is passed to the template from the Javascript code and is automatically replaced when the template is rendered. On line 3, using the same principle, a `text` variable is displayed. The standard tags used by JsRender are `{}{}`. They are used to identify JsRender code. However they clash with Jinja’s tags and cannot be used as errors occur in Flask. Fortunately, JsRender provides a function to change the tags used. The tags were changed to `<%` and `%>`

The templates defined in `chatbot.html` are rendered and used in the JavaScript program. Listing 13 shows an example of rendering the message template. Since two variables are used in the template, `title` and `bg_col`, they have to be passed when rendering it. This is done by creating the JavaScript object called `data` on line 1. The message text is stored in it as well as the necessary background colour which is set depending on whether the user or the bot has sent the message. On line 6 the template is loaded. JsRender can be used as an extension to JQuery by default, which is why the global JQuery variable `$` is available. The template is rendered on line 7. It is required that any data is passed during this step. The result is the rendered HTML with the variable values substituted. The result is stored in a variable. It can then be added to the chatbox using a JQuery selector. The `append()` function is called on the selector to add the resulting HTML to the chatbox. This method does not require reloading of the page and it changes the content dynamically.

Listing 13: Rendering the JsRender message template.

```
1 const data = {
```

```
2 "text": text ,  
3 "bg_col": user ? "info" : "dark"  
4 };  
5  
6 const messageTmpl = $.templates('#messageTmpl');  
7 const messageHtml = messageTmpl.render(data);
```

JsRender is also used for the loading animation and the error messages shown to the user when they haven't given the program any input. The loading animation is pre-rendered as it does not have any dynamic content. The error message template utilises a Bootstrap alert which provides a coloured box which can be dismissed by clicking an "X" button in its top right corner.

4.5 Back-end and Front-end Communication

The communication between the front-end and the back-end is required as the user input has to be sent to the back-end program for processing. The resulting output has to be sent to the front-end. This communication was achieved with the JavaScript.

4.5.1 Requests

The JavaScript program forwards user questions to the back-end. Using JQuery event handlers, it is waited for a click on the Send button or the Enter keyboard key to be pressed. When either of these events occurs, the text stored in the text input field validated and passed to another method. If the user hasn't entered any text, they are shown an error message. After it has been confirmed the input is valid, a GET request is sent to the server (the back-end program). The data is sent in the JSON format. Example JSON data can be seen in Listing 14. Data passing can occur in the background without affecting the responsiveness of the page since JavaScript supports asynchronous requests. To let the user know that the program is processing their input, a loading animation is displayed. When the request is completed and a response has been returned by the back-end program, the loading animation is hidden and the bot response is added to the chatbox.

Listing 14: Example data sent to the back-end for processing in the JSON format.

```
1 {
2     "request_type": "regular",
3     "message": "Hi, how are you?"
4 }
```

4.6 Extra Features

Three extra features were implemented to improve the functionality of the application.

4.6.1 Command line Interface

During development and testing, certain repetitive tasks had to be done manually. For instance, to collect training data, the “crawler” Python script had to be run every time. Since it is a separate program, it had to be run manually. If any of its parameters were changed, e.g. how many pages are crawled or how many threads are used, its code had to be modified. For this reason, a command line interface was implemented so that a central place exists from where repeated actions could be run.

Flask provides an interface for implementing custom commands. All Flask commands start with “flask” followed by the command name. For example, the command to run the Flask server, which can be executed from a terminal, is “flask run”. Optional parameters can be set using parameter switches. An example is running the Flask server with multiple threads with the command “flask run –with-threads”. Flask implements this functionality with the Python module Click.

The custom commands implemented are:

- **flask crawl** – runs the crawler Python script to collect data from Stack Overflow. Optional parameters are:
 - ◊ **-t, --threads** – how many threads should be used when crawling.
 - ◊ **-p, --pages** – how many pages should each thread crawl.
 - ◊ **-v, --verbose** – prints detailed command line output of what is currently happening.

- **flask train** – trains the chatbot with the training data collected with “flask crawl”.
- **flask clean** – deletes the training data collected with “flask crawl”. Optional parameters are:
 - ◊ **-y, --yes** – automatically deletes the files without asking the user for confirmation.
- **flask del_db** – deletes the database file for the chatbot. Optional parameters are:
 - ◊ **-y, --yes** – automatically deletes the database without asking the user for confirmation.

4.6.2 Alternate Responses

Stack Overflow threads usually contain more than one answer to a question. Furthermore, in programming, usually there are multiple ways to achieve something depending on the situation. This is why the web crawler by default collects more than one answer to a question. However, when the user asks the chatbot a question, by default the most upvoted response is selected. There is no way to access the other collected data. For this reason, an extra feature was implemented to allow the user to request an alternate response in the event that the original response wasn’t satisfactory.

The initial idea for the feature was to add a button to each response. If the button is clicked, another response would be given to the user. However, it was decided that a simpler implementation might be better. Since the user can “talk” to the chatbot, simply asking it to provide an “alternate response” would make the experience of communicating with the chatbot more authentic. The user would feel like they are communication with another person.

Several approaches to implementing this feature existed, however ultimately it was decided that the default BestMatch logic adapter should be modified. Normally, the BestMatch adapter generates two lists of responses – the *response list* and the *alternate response list*. The response list is first checked, and if one of the responses in it has a confidence value greater than a previously set threshold, it is automatically accepted as the best answer. If none of the responses has a high value of confidence, the alternate response list is created using different response selection parameters. Since these lists are generated every time a question is asked, they can be stored on the server.

Therefore, there is always a list of possible alternate responses to the last question the user asked.

The logic adapter returns the first response in the response list (index 0 of the list) by default as the list is ordered by confidence. If an alternate response is requested, a response from the cached list can be returned. This is achieved by calling the `pop()` method on the list of cached responses. It returns the item stored at the index given as a parameter and removes it from the list. Therefore, by calling `pop(1)`, an alternate response will be acquired as long as there are items left in the list. It is also checked whether a result can be returned from this index. If not, a message is returned instead to inform the user.

The logic adapter runs on the server, however client-side code is also required. In the chatbot Javascript program, an additional function was added to handle alternate responses. Every time the user provides input, it is checked if it matches the phrase “alternate response”. If it does, the function `getAlternateResponse()` is called with the input. Otherwise, the input is forwarded to the default `getBotResponse()` function. The function which provides alternate response works similarly to the regular response function. The main difference is the request it sends to the back-end. Since the request asks the server to perform a different action, its body contains extra information. To distinguish it from regular response requests, the value of the `request_type` variable is set to “alternate” (to signify that an “alternate” response is requested) as opposed to “regular” (which signifies a regular response).

The back-end function which processes the requests was also extended. When a request of type “alternate” is received, a response is requested from the bot. However, instead of asking for a response of a specific question, the command “ALT_RESPONSE” is set as the question. The BestMatch logic adapter then checks whether the question it is asked starts with the command. If it does, then the process described above is carried out.

4.6.3 Feedback

During the testing process of the application, it was discovered that the default response to some questions was not always the best response. For this reason, another feature was implemented: user feedback. It would allow the user to “rate” a response. Better answers would receive higher rating. Thus, whenever new users use the application, they would receive answers with the highest rating.

The first step to implementing this feature was adding a voting or rating functionality to the front-end. The initial idea was to implement a star-rating system where 5 stars would indicate that the answer was very good, and 1 star would indicate that it was not coherent. However, a simpler to use system was implemented. Each time the user receives a response from the chatbot, the bottom part of the message box would contain a question to the user: “Was this answer helpful?”. The user would then be able to click on a “Yes” or “No” button to give feedback on the appropriateness of the answer.

Implementing the feedback feature in the front-end side of the program required the creation of another JsRender template. The template contains the HTML code for the question and the two buttons. The text and the buttons do not need to change dynamically; however, they need to be uniquely identifiable. This is necessary as when one of the buttons is pressed, the question it refers to has to be identified. To implement this functionality, the feedback panel was placed inside a Bootstrap `row`. Each of these rows was given a unique `id` in the format `row-id-<ID>`, where `<ID>` refers to a unique integer. The code used to create the template can be seen in Listing 15.

Listing 15: The JsRender template used to generate HTML code for a feedback panel.

```
1 <script id="feedbackTmpl" type="text/x-jsrender">
2 <hr/>
3 <div class="row row-id-<%=btn_id%>">
4   ...
5 </script>
```

In the JavaScript code, when rendering the template, a value for the variable `btn_id` is passed to the template. The value is a unique integer which is incremented every time a feedback template is rendered. The same variable is used to identify each of the buttons. Additionally, each button has an `onClick` HTML property. In it, a function defined in the JavaScript program is called: the `getFeedback()` function. It takes 2 parameters: the unique ID of the button that was clicked and a string. The second parameter contains “yes” for the “Yes” button and “no” for the “No” button.

When `getFeedback()` is called, it processes the data passed to it. The ID value is used to find information about the answer the user wishes to rate. For this reason, all answers given to the user have to be stored. This can be done on the server-side, however, since JavaScript is a client-side language, the server-side history would contain data about all the answers sent to all

users using the website. If there are a lot of users, searching through this data would take a lot of time and resources. For this reason, the decision to store this data on the client-side was made. Every time the user asks a question, the received response is stored in a JavaScript array. Incidentally, the position of each answer in the array corresponds to the unique ID of each of the buttons. Thus, the `getFeedback()` function can easily find the response in the array of all responses by selecting the item at index ID of the array. A request can then be made to server. The request body contains the following information: the type of the request (set to “feedback”) so that the request can be identified, the rating the user gave to the question (+1 or -1) and the answer itself.

When the server receives a request of type “feedback”, it extracts all the data from it. The data is then passed the `bot.py` script. A function checks whether the feedback is “yes” or “no”. If the value is “yes”, then the function in the storage adapter `update_rating()` is called with a parameter of 1, if it’s “no”, then -1 is passed to the same function. In order to find the response in the database, the text of the response is passed to the same function. The database is then searched using an SQL query created with the SQLAlchemy library. The query selects a statement from the “statements” table the text of which is the same as the text passed to the function. The rating value of the selected statement can then be updated. Lastly, the changes are committed to the database.

4.7 Summary

Fig. 11 illustrates the process carried out when a request is received by the back-end program.

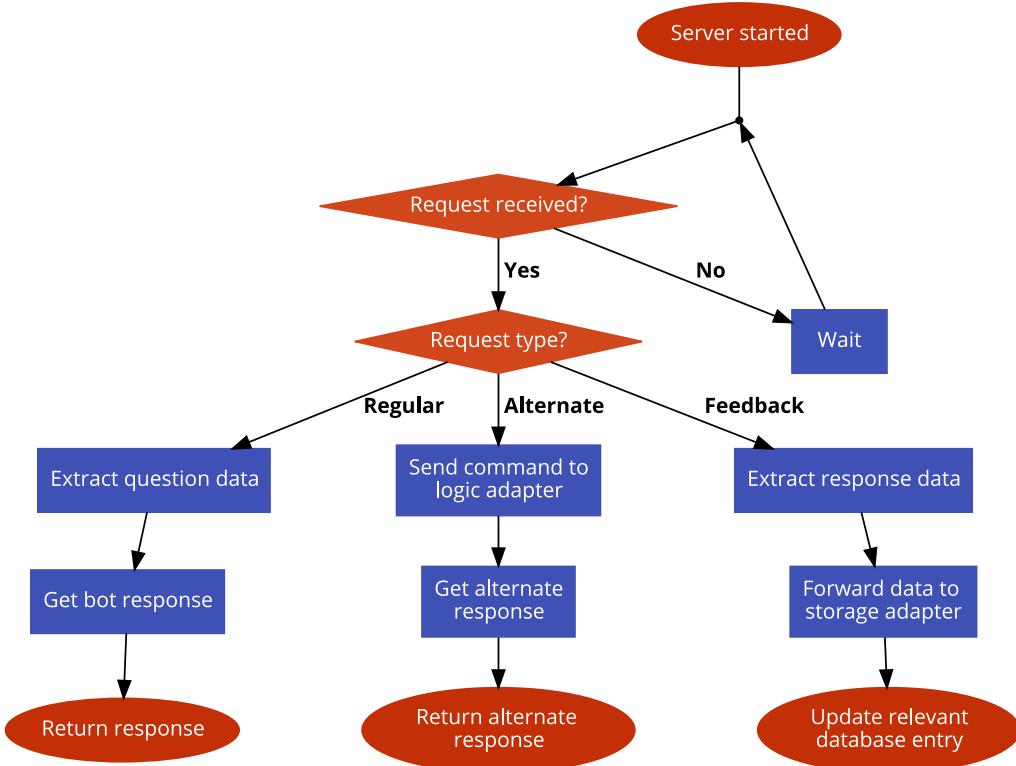


Figure 11: A summary of request processing in the back-end.

5 Evaluation

The aim of this project was to investigate whether chatbots can be utilised in education. To achieve this goal, a prototype of a chatbot was created. The requirements set before the development started were as follows:

- The application should produce coherent output when given user input.
- The application should utilise Natural Language Processing techniques to “understand” user input.
- The application should utilise Natural Language Processing to compare user input to known data.
- The application should have a user-friendly user interface.
- The application should be able to learn and improve through its interactions with users.
- The application should be easily trainable with additional data whenever necessary.
- The application should be accessible from both mobile and desktop computing devices.
- Etc?

This section will evaluate the implemented prototype. Two approaches will be taken: it will be compared to similar applications and a group of users will test it and provide feedback.

5.1 Preparation

To prepare the application for testing, firstly data was collected. The crawler program described in Section 4.3.1 was used with the parameters summarised in Table 2. To speed up the collection process, 4 threads were utilised. Each thread was assigned the task of crawling 10 pages of questions. The resulting data contained information about 600 questions. For each question, the 3 top answers were collected (where applicable, questions may have less answers). The resulting data contained approximately 1800 answers to 600 questions.

Even though it would have been beneficial to collect more data, Stack Overflow limit their incoming traffic. If a user sends too many requests to their servers, their access to the website is blocked. This proved to be a limitation

Table 2: Parameters used for the web crawler when collecting data in preparation for testing.

Parameter	Value
Number of threads	4
Number of pages crawled by each thread	10
Number of questions per page	15
Number of answers per question	3
Total questions collected	600
Total answers collected	1800

in the data collection stage of the evaluation process. For this reason, the collected data was deemed satisfactory for testing purposes.

Secondly, the chatbot was trained using this data. In addition, the Python module “chatterbot-corpus” was also used for training. It provides example conversational data. In this case the data labelled “greetings” was used so that the chatbot can greet the user when input similar to “hi” and “how are you?” is provided. Chatterbot-corpus includes several other datasets, e.g. “jokes” and “trivia”. However, these were not used as the purpose of this chatbot is to answer specific programming questions.

5.2 Comparison with Cleverbot

Cleverbot (Carpenter, 2019) is one of the most popular chatbots which currently exist. It was developed by Rollo Carpenter as a successor to the Jabberwacky chatbot. It was initially created in 1988 and was made public in 2006. It does not use machine learning to improve itself. Instead, a pattern matching technique is used to compare user input “fuzzily” to stored data from past conversations (Carpenter, 2019).

Input correction

Whenever Cleverbot is given user input, it corrects the punctuation and the word case if necessary. For instance, input such as “hi how are you” is converted to “Hi how are you.”. Commas and other punctuation marks

will not be automatically inserted. It is also assumed that a full-stop should be added to the end of the sentence if it's missing. The reason for this functionality is probably to allow Cleverbot to match the input to the data it uses to generate a response.

While this approach might be beneficial, it does not work for the application developed for this project. A similar feature was tested by adding the JavaScript library Typopo. It can check text for common mistakes such as extra whitespace, extra dashes, wrong word case, etc. and correct it. The result, however, was not satisfactory. In programming, a lot of combinations of symbols are used which might be considered wrong by a spell-checking program. For instance, the question "what is the --> operator used for" is corrected automatically to "What is the -> operator used for". The library matches the input to known patterns, in this case the two dashes, and it removes one of them. This, however, changes the question asked by the user. If a "->" operator existed, a response related to it will be returned. Ultimately, this input correction functionality was not implemented. Furthermore, it is not required since punctuation and letter case do not affect the generated response since the chatbot does not utilise pattern matching.

Context

Cleverbot can have longer conversations with a user. For instance, consider the following conversation:

```
User: Tell me a joke.  
Cleverbot: Knock Knock.  
User: Who is there?  
Cleverbot: Broken pencil.  
User: Broken pencil who?  
Cleverbot: Never mind it's pointless.
```

The user asks for a joke and Cleverbot replies with the beginning of a "knock, knock" joke. The user then asks the bot for the rest of the joke twice and Cleverbot responds accordingly. To ensure that this is not random output of a statement, the joke was found online. Another version of the joke is as follows:

```
A: Knock, knock!  
B: Who's there?  
A: Broken pencil.  
B: Broken pencil who?  
A: Forget it - this joke is pointless.
```

Through this interaction, Cleverbot appears to understand the context of the conversation. It is able to continue talking about a previously set topic. However, it is assumed that this possible due to the large amount of data Cleverbot has access to. A similar conversation with another user probably took place in the past and the user provided similar input to the bot's statements. It is confirmed that Cleverbot has no knowledge of context by continuing the conversation:

[...]

Cleverbot: Never mind it's pointless.

User: Why is it pointless?

Cleverbot: Because I already know that.

User: What do you know?

Cleverbot: You're not too clever.

In this case, the chatbot does not know how to continue the conversation and a response is selected from another conversation in its database.

On the other hand, the application developed for this project neither has any knowledge of context nor tries to imitate it. For instance, when asked the question "what are move semantics?", the bot replies successfully. However, if the user tried to continue the conversation by asking "when to use them?" to bot is unable to generate a response. It should be noted that if it had data about a question similar to "when to use them?", the response to that question will be returned to the user. Therefore, the chatbot is unable keep track of previous statements in the conversation. Such a feature would be extremely beneficial; however, implementing it would not be a trivial task and such a system with context awareness has yet to be created. A similar approach to Cleverbot could be taken where whole conversations are stored. However, Cleverbot is able to learn from conversations with its users since it does not have a specific domain of knowledge. On the other hand, this chatbot's purpose is to teach students by giving them answers it already knows. For this reason, it would be unable to learn from user input. Additionally, because of this realisation, the chatbot was made "read-only" which does not allow it to store conversations with its users.

Paraphrased input

Cleverbot can answer the same question worded differently. For example, if the user asks "what is your favourite colour?", Cleverbot replies "Don't have one.". If the user then asks "what colour do you like the most?", the bot replies with "Blue.". Even though the answers to both questions do not match, they are coherent and are linked to the question. However, this

is probably possible due to the amount of data Cleverbot has access to. Nevertheless, Cleverbot can process paraphrased user input.

On the other hand, the chatbot developed for this project cannot respond to paraphrased input. For example, when asked “how are you?”, it replies with “Very well, thanks”. If the question is changed to “how is it going?”, it responds with “Good.”. This is possible because the chatbot database contains responses for both questions. In reality, the chatbot is not necessarily able to realise both questions have very similar meaning. Furthermore, when asked “what is the difference between a pointer and a reference?”, the chatbot generates a correct answer. Analysing the training data shows that the statement the chatbot “knows” is actually “What are the differences between a pointer variable and a reference variable in C++?”. Therefore, it is able to process partially paraphrased input. However, it should be noted that most of the words which convey the meaning of the sentence are present in the paraphrased question. Thus, it should not be surprising that the results are good. Unfortunately, when the question is paraphrased drastically to, for instance, “references vs. pointers”, the chatbot replies with a known reply to the statement “When to use references vs. pointers”. While the information might be useful for the user, their intention was to find out is the differences between the two are, not when to use each one.

Unfortunately, no easy solution to this problem exists. Making the bot “smarter” would require developing a more complex system. Furthermore, using a premade library would be impossible. Instead, the system would have to be written for this specific purpose. However, a suggestion for improvement of this chatbot is given: after the training data is collected, another program could read each question, and generate paraphrased questions from it. The answers of the original questions can then also be associated with the paraphrased versions. This would be a naïve solution, however it might produce satisfactory results for the purposes of this project.

5.3 User Testing

A group of users was asked to provide feedback for the application twice. The first time was after completing the initial prototype of the website described in Section 4. Four users were asked to test the application and provide general feedback. The aim was to find out whether the application was user-friendly and worked as expected. The second testing session was after some changes were implemented based on the feedback from the first session. A larger number of users were asked to participate.

5.3.1 First Testing Session

The first testing session took place after an initial prototype of the application was built. All features described in Section 4 were implemented and could be used. Five users participated in the session. All of them were Computing students in their final year at university. All of them had advanced overall knowledge of programming and at least intermediate knowledge of C++. The aim of this session was to identify user interface problems so that the application can be made more user-friendly. Additionally, another objective was to find bugs in back-end or the front-end. The users were not asked any specific questions and were able to provide feedback in the form of notes without any specific format.

Appendix E illustrates the result of the first testing session. The column named “Feedback” lists the points the testers made about what they thought should be changed. The column “Changes made” describes what changes were implemented based on the user feedback. It should be noted that the results table only shows the negative feedback received. A more detailed summary of the problems and their solutions follows.

All testers agreed that the style of hyperlinks was not satisfactory. One of the testers made the point that it looked like a “grammatical or spelling error which is underlined by a spell-checking program”. This is caused by the initial style that was implemented: the text colour of hyperlinks was white (the same as regular text), and they were underlined with a red line. Changing the colour to yellow and green was tested, however, the same tester noted that it still looked like a “grammatical error” due to these colours being associated with errors (red), warnings (yellow) or correctness (green). Another tester suggested using a different colour for the hyperlink text and underlining it with a line of the same colour. In the end, this idea was implemented, and the colour of hyperlink text and underlining was changed to light blue. As a result, the testers agreed that it looked better and it was clear which words were a hyperlink.

Another problem caused by the style of hyperlinks was that the “help” link in the initial chatbot message was not visible. The testers thought it was emphasized but did not realise they can click it. Additionally, they did not realise they can use the “help” command by typing “help” as a message in the chat. This was caused by their inability to access the help message which could only be seen by clicking on the “help” link. To rectify this problem, an additional “help” button was added at the top of the page. Furthermore, the initial chatbot message was changed to specifically tell the user they

can either type “help” in the chat, click the “help” link or click the button at the top of the page. As a result, the testers agreed that this is more user-friendly.

One tester made the point that typing the “alternate response” command can become tedious when it is done repeatedly. They suggested adding a button. The button was added to the bottom of every bot message (in the panel for feedback). After testing the changes, the testers agreed that while this solution worked, the button was only accessible if the user hasn’t clicked on the feedback buttons. If one of them is clicked, the contents of the panel change. A suggestion to add the button to the message shown when the user clicks the “no” feedback button was made. Since clicking the “no” button means that the user was not satisfied with the response, it would make sense to add the “alternate response” button to the new message shown. This change was also implemented, and the testers agreed that the result was satisfactory.

One of the users who tested the website on a desktop computer with a large screen noted that the webpage fills their entire browser window. While this is to be expected as the responsive design of the website ensures that the content’s size adapts based on the screen, this caused a problem. The tester said that this makes it very difficult to read long answers as they look like a “wall of text”. After researching what other websites with a lot of text do to fix this problem, it was realised that very rarely websites use the full width of the screen. Usually, columns on the left and right are used for other content. The text takes up about 60% of the width of the screen. Additionally, while online chatrooms often used the full width of the browser window, the text messages rarely take up that much space. As a result, the width of the Bootstrap Container was reduced. This change only affected devices with big screens. No changes were made for mobile devices. The testers agreed that it was easier to read longer responses.

Other smaller changes made based on the feedback were:

- The help message was changed to more clearly indicate what features the user has access to and how to use them.
- Another webpage (“about this project”) was added to explain the purpose of the website and to give credit to Stack Overflow for the data collected from there.
- A navigation bar (menu) was added to the top of the page with a “Home” and “About” buttons which redirect to the respective pages. The “Help” button was also placed in the menu. Clicking on it, simply

writes “help” as a question on behalf of the user. The testers agreed they liked this behaviour since it teaches the user how to use text commands. For this reason, the functionality was kept.

- One of the testers discovered that while waiting for a response, more questions can be asked which results in the inability of the bot to respond. To fix this problem, the “Send” button was disabled while waiting for a response as well as pressing the “Enter” keyboard key.

Lastly, the testers agreed that the bot was able to respond to about half of their questions. However, this was to be expected since the training data used was of limited quantity.

The improved website can be seen in Fig. 12.

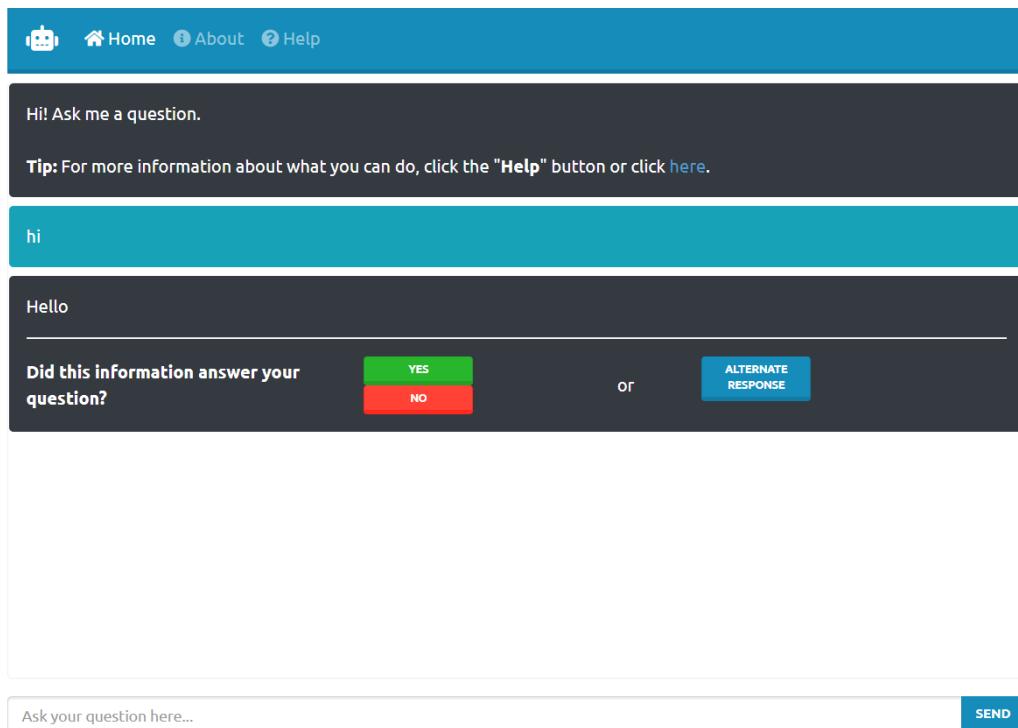


Figure 12: The improved user interface.

5.3.2 Second Testing Session

For the second testing session, the same data collected in Section 5.1 was used. A second website was hosted on PythonAnywhere so that the users can access it from their own devices. The development branch of the repos-

itory was hosted as it was unclear whether the changes made after the first testing session could be considered stable.

The aim of the session was to test the improved prototype with as many users as possible. In the first testing session the participants had more advanced knowledge of programming. However, in the second session the objective was to have the application tested by beginner programmers. Since the target audience for the project is new university students, it was estimated that the results would be more accurate with this approach.

For the testing session a questionnaire was prepared using the website SurveyGizmo. It was selected as it provided a free account plan which does not limit the questions per survey. The final questions can be seen in Appendix F.

Each tester was sent a link to the survey and a link to the newly set up website. They were given a short description of the aim of the project, and were informed that the chatbot can answer specific C++ questions. Each participant was also instructed to test the application and then complete the survey. Twelve testers agreed to take part.

Results

A bar chart of the responses to question 1 can be seen in Fig. 13. The results showed that 50% of the participants were beginner programmers. This result is satisfactory as feedback from users with different knowledge levels will be available while most of them are new programmers.

The results for question 2 can be seen in Fig. 14, The results were positive, and they proved that the changes made after the first testing session were favourable.

Question 5 asked the users if they were able to use the alternate response feature. The results show that 16.67% answered that they neither agree nor disagree and 8.33% that they disagree. When asked what prevented them from using the feature, their feedback was that whenever the chatbot answered with an incorrect response, the answer produced by the alternate response feature would be an answer to the same wrong question. This behaviour, however, is to be expected since the alternate response feature selects another response from the previously generated list of responses. To improve this feature, it would be required to regenerate the list of responses every time the user requests an alternate response instead of storing it the first time they ask a question. The results are illustrated in Fig. 15.

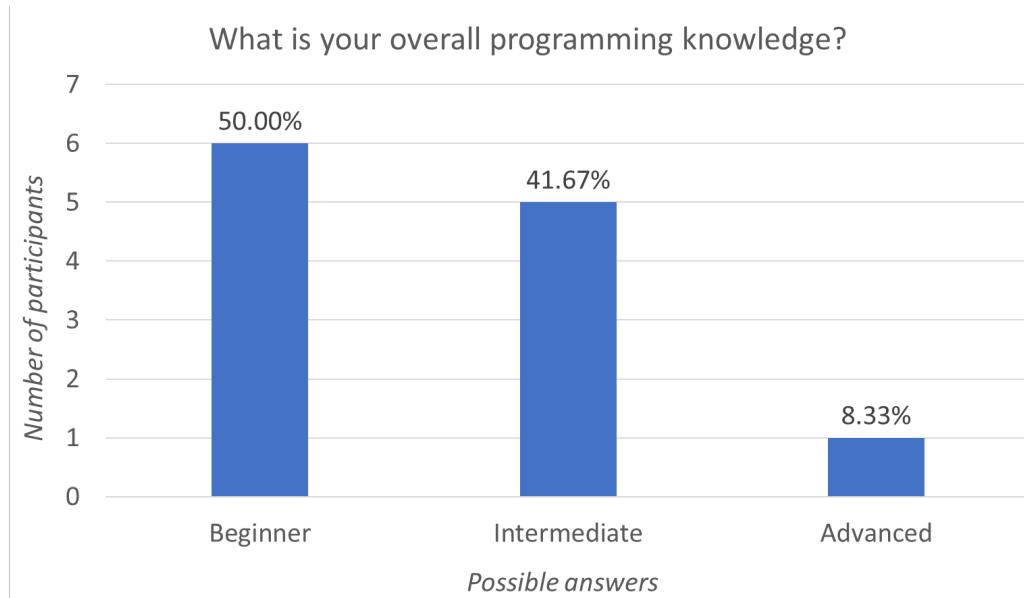


Figure 13: Bar chart showing the overall programming knowledge level of the participants in the survey. The y-axis corresponds to the number of participants who selected an answer, while the x-axis represents each possible answer.

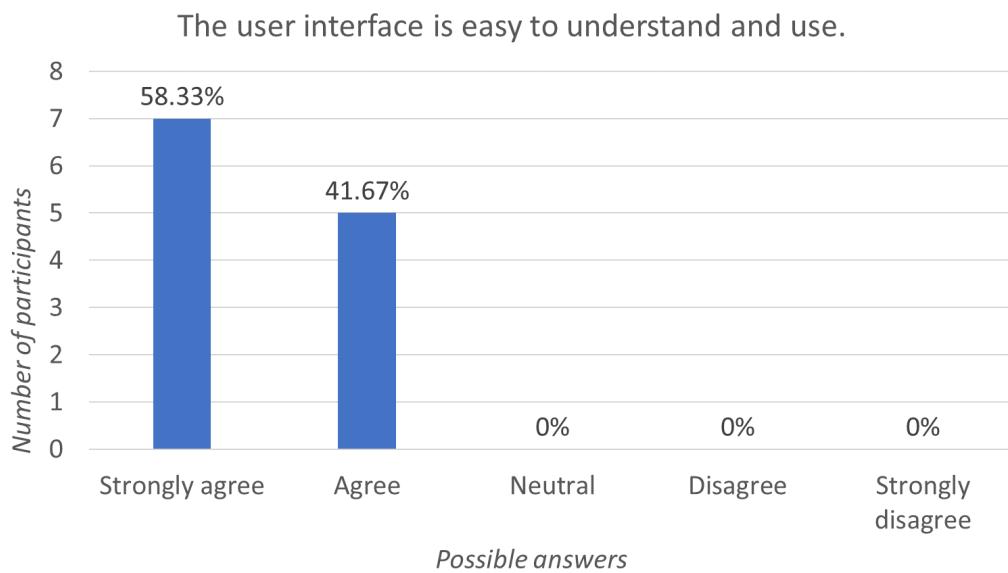


Figure 14: Bar chart showing the responses to question 2. The y-axis corresponds to the number of participants who selected an answer, while the x-axis represents each possible answer.

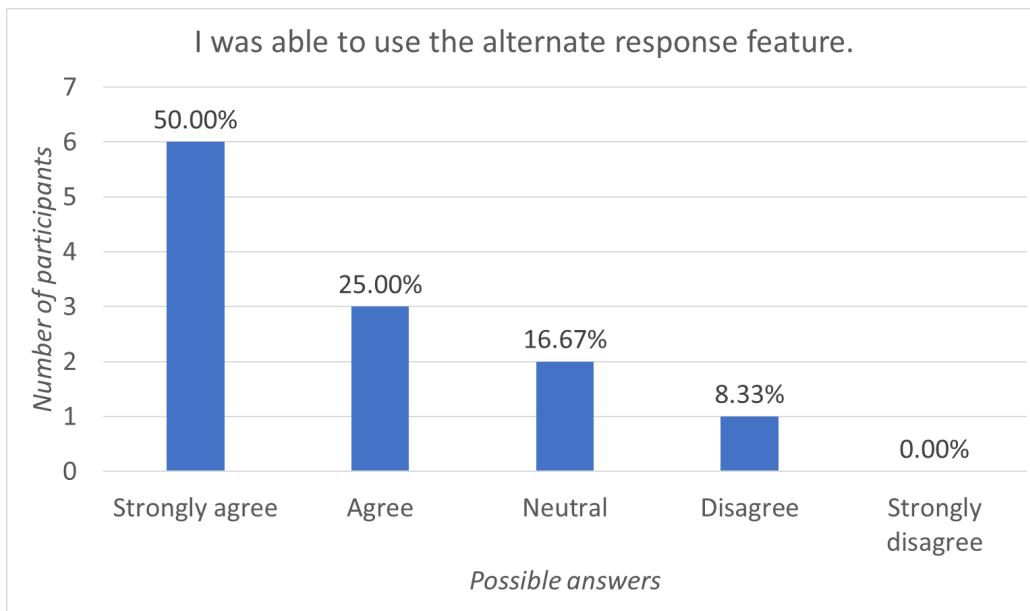


Figure 15: Bar chart showing the responses to question 5. The y-axis corresponds to the number of participants who selected an answer, while the x-axis represents each possible answer.

The results for question 6 show that 8.33% neither agreed nor disagreed. When asked why the selected this answer, they replied that they did not understand how this feature affected the future responses of the chatbot. For this reason, a consideration was made to add an explanation in the help message since it would motivate the users to use the feature. As a result, the chatbot will improve more. The results for this question can be seen in Fig. 16.

The answers for question 7 were positive (“appropriate error messages are shown”). The users responded that they strongly agree (58.33%) and that they agree (41.67%).

The results for question 8 were not considered as it was realised that the free hosting option on PythonAnywhere limits the speed of operation of the application. However, the results were positive probably due to the small amount of users using the website concurrently.

Question 9 asked the user whether the chatbot answered their questions accurately. 16.67% answered that they neither agree nor disagree. When asked why they chose this answer, the participants said that the chatbot was unable to answer some of their questions. This was a result of the

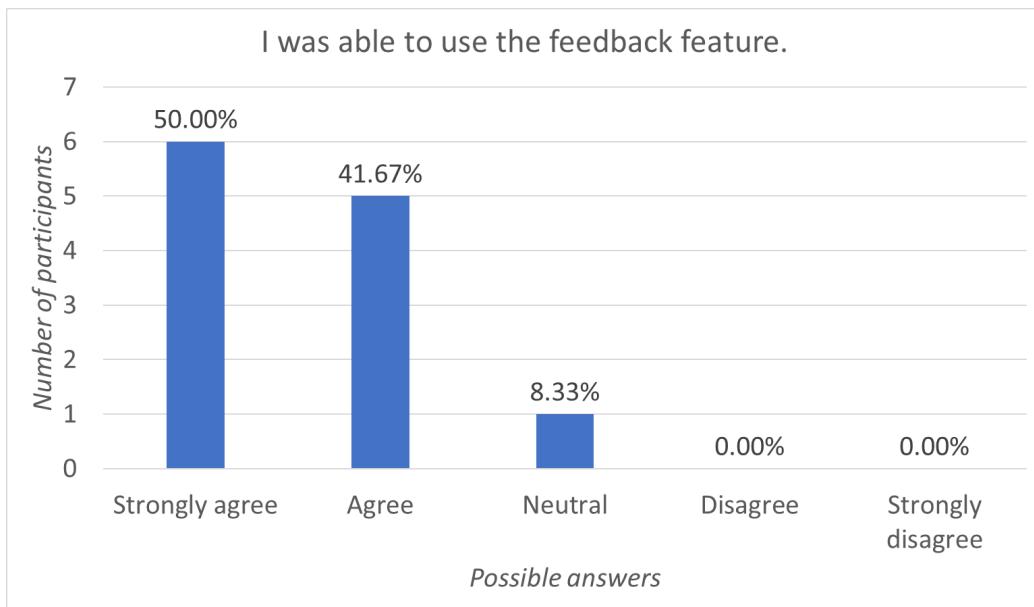


Figure 16: Bar chart showing the responses to question 6. The y-axis corresponds to the number of participants who selected an answer, while the x-axis represents each possible answer.

limited amount of training data. The answers to question 9 are summarised in Fig. 17.

The feedback for question 10 was positive: 66.67% of the testers strongly agreed and 33.33% agreed that they were overall satisfied with the application. As a conclusion, it can be said that the participants approved of the prototype since most of the responses were positive.

Most of the testers did not provide any additional feedback in the last question. One made the point that the layout of the website did not work on their smartphone (iPhone). The webpage did not fit in their browser window. Another piece of feedback was that the chatbot does not respond to questions which include symbols such as &, [,], | . However, when trying to find out what caused that problem, no issues were found with these symbols. Another tester mentioned that the chatbot could not respond to general questions such as “what is C++?” and “what is the difference between C++ and other languages”. Even though the purpose of the chatbot was to help with specific problems, such questions are beneficial information for new programmers. Additional data was collected from the official website of the C++ standard (Standard CPP Foundation, 2019) by creating another crawler program (see Appendix G).

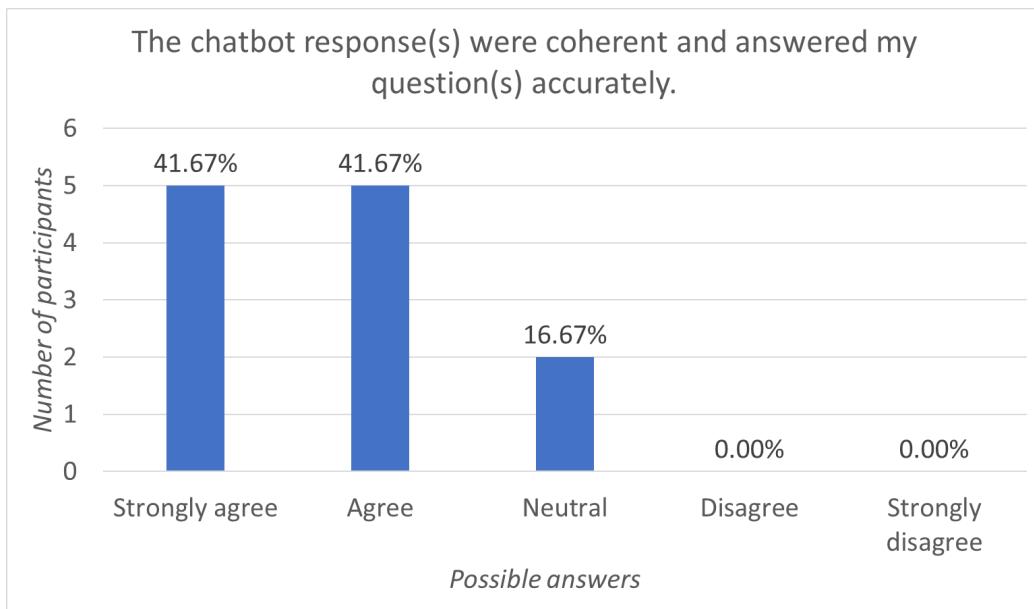


Figure 17: Bar chart showing the responses to question 9. The y-axis corresponds to the number of participants who selected an answer, while the x-axis represents each possible answer.

Lastly, one of the testers noticed that there was a problem with the feedback feature which only occurred for some responses. An example is sending the message “hello” to the bot. The initial response is “Hi”. When the feedback feature is used to downvote the answer, the bot chooses the next response (“Greetings!”). However, if this answer is voted negatively, the bot does not return the initial answer (“Hi”). It was discovered that this is caused by how the data is stored into the database. When updating the rating of the response, the search for an entry with text “Greetings” finds a result with an empty “in_response_to” value. This means this value is never selected from the database when the user asks a question. To fix this problem, an additional requirement was added when selecting an entry from the database: it is checked whether the “in_response_to” field is empty.

A summary of the feedback from the survey is presented in Table 3. The percentage of “strongly agree” responses was 57.4% of the total responses. The users selected “agree” for 36.1% of the responses, 4.6% of the responses were “Neutral” and 1.9% were “Disagree”. It can be concluded that the feedback was very positive even though some issues were highlighted.

Overall, it can be concluded that the testers were satisfied with the application. Since this is an initial prototype, problems could be expected, how-

Table 3: A summary of the results from the questionnaire.

Question	Strongly agree	Agree	Neutral	Disagree	Strongly disagree
The user interface is easy to understand and use.	7	5	0	0	0
The user interface is visually pleasing.	6	5	0	1	0
I was able to understand how to use all the features of the application.	8	4	0	0	0
I was able to use the alternate response feature.	6	3	2	1	0
I was able to use the feedback feature.	6	5	1	0	0
Appropriate error messages are shown.	7	5	0	0	0
The chatbot answered my question(s) quickly.	9	3	0	0	0
The chatbot response(s) answered my question(s) accurately.	5	5	2	0	0
I am overall satisfied with the functionality of the application.	8	4	0	0	0
Total	62	39	5	2	0

ever some issues could have been avoided by simply collecting more training data.

6 Conclusions

The aim of the project was to investigate whether chatbots can be used as a tool in education by implementing a suitable application and evaluating it.

A review of academic literature was conducted to highlight some of the key technologies and concepts related to the development of a natural language generation systems. In addition, similar existing applications were investigated so that the approaches to their development could be compared. The results of the review were used during the development of a prototype of the proposed software.

The initial objectives of the project were met. A system which can successfully be trained with data about a desired topic was implemented. Furthermore, the system was made easily accessible since it provides an online interface which adapts to the screen size of the device used. The interface design decisions made were based on existing messaging applications which have proven to be user-friendly. In addition, extra features were added which surpassed the initial plan. Lastly, the configuration process of the system was simplified so that an administrator could configure and use it easily.

The project was evaluated by comparison with a similar application. The results showed that the rivaling program was more advanced, however its superiority can be mostly attributed to the large amounts of training data it has access to, not the response selection method. Moreover, the system developed for this project was evaluated by a group of users which resembles its expected target audience. The results were overwhelmingly positive, even though issues were discovered. The participants were overall satisfied with the system.

The areas where the project did not meet its objectives were its ability understand natural language. While the deliverable utilises Natural Language Processing techniques to process input and generate output, the algorithm is limited. This is mainly due to the library selected for the implementation of the back-end. The chatbot can answer questions, however training data of high quality and large quantity would be required for more advanced use. There were factors which prevented the collection of more data in this study, however the web crawling program that was developed would be able to achieve this if there were no outside restrictions. Other issues highlighted by the testers were problems with the responsive design and the inability of the application to answer paraphrased questions.

To conclude, it can be said that the delivered chatbot could find use. It would be impossible for it to provide as much knowledge as a teacher, however it can be used by a teacher to provide ready access factual data.

7 Future Work

While the project met most of its requirements, its functionality can be improved as the evaluation process showed. The following suggestions for improvements and additions are made:

- The alternate response feature can only retrieve previously stored responses. While this approach works, if the chatbot failed to identify the question initially, the alternate responses returned would be incorrect as well. A solution to this problem would be searching through the database and generating a new list of responses using different parameters when the user request an alternate response. This solution would be slower, however the responses would be more accurate.
- The chatbot should be able to understand paraphrased input. At present, it is able to do so in some cases. However, improving this system would be crucial since it would be able to respond to more questions and provide more assistance.
- The user interface is not compatible with some mobile devices as some of the testers discovered. It should be improved since it is expected that it will be used often. The participants of the survey also confirmed that they prefer using the application on a mobile device.
- An online retrieval feature could be added in the event that the chatbot does not have information about a question. It could perform a web search and display relevant URLs to the user.
- An administrator page could be added where the chatbot can be configured instead of using the command line interface. Furthermore, statistics could be displayed about the number of questions the chatbot has answered and an interface for adding more training data could be implemented.

References

- Carbone, M. (2016). Remembering SmarterChild. Retrieved November 12, 2018, from <https://blog.talla.com/remembering-smarterchild>
- Carpenter, R. (1997). Jabberwacky.com. Retrieved October 26, 2018, from <http://www.jabberwacky.com/>
- Carpenter, R. (2019). Cleverbot. Retrieved March 12, 2019, from <https://www.cleverbot.com/>
- Colby, K. (1981). Modeling a paranoid mind. *Behavioral and Brain Sciences*, 4, 515–534.
- Flask. (2019). Official Flask Documentation. Retrieved March 28, 2019, from <http://flask.pocoo.org/docs/1.0/>
- Floridi, L., Taddeo, M., & Turilli, M. (2009). Turing's imitation game: Still an impossible challenge for all machines and some judges - An evaluation of the 2008 loebner contest. *Minds and Machines*, 19(1), 145–150.
- Gabbatt, A. (2011). IBM computer Watson wins Jeopardy clash. Retrieved December 11, 2018, from <https://www.theguardian.com/technology/2011/feb/17/ibm-computer-watson-wins-jeopardy>
- Goldberg, Y. (2015). A Primer on Neural Network Models for Natural Language Processing. *Report*, 57, 1–75.
- Goldberg, Y., & Hirst, G. (2017). *Neural network methods in natural language processing*. Morgan & Claypool Publishers.
- Griffin, A. (2016). Tay tweets: Microsoft creates bizarre twitter robot for people to chat to. Retrieved November 12, 2018, from <https://www.independent.co.uk/life-style/gadgets-and-tech/news/tay-tweets-microsoft-creates-bizarre-twitter-bot-for-people-to-chat-to-a6947806.html>
- Gunther Cox. (2019). ChatterBot 1.0.2 documentation. Retrieved October 2, 2018, from <https://chatterbot.readthedocs.io/en/stable/>

Intelligence. (n.d.), In *Oxford English Dictionary Online*. Oxford University Press. Retrieved October 24, 2018, from <https://en.oxforddictionaries.com/definition/intelligence>

Kononenko, I., & Kukar, M. (2007). *Machine learning and data mining: Introduction to principles and algorithms*. Horwood Publishing Limited.

Kotsiantis, S. B. (2007). Supervised machine learning: A review of classification techniques.

Loebner.net. (2017). What is the Loebner Prize? Retrieved October 15, 2018, from <https://web.archive.org/web/20170330123946/http://www.loebner.net/Prizef/loebner-prize.html>

Lovins, J. B. (1968). Development of a stemming algorithm. *Mech. Translat. & Comp. Linguistics*, 11, 22–31.

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to information retrieval*. New York, NY, USA: Cambridge University Press.

Martinez, A. R. (2010). Natural language processing. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(3), 352–357.

McCallum, A., Nigam, K. et al. (1998). A comparison of event models for naive bayes text classification. In *Aaai-98 workshop on learning for text categorization* (Vol. 752, 1, pp. 41–48). Citeseer.

Michael L. Mauldin. (1994). Chatterbots, TinyMuds, and the Turing test: Entering the Loebner Prize competition. *Aaai*, (4), 254.

Paice, C. D. (1990). Another stemmer. *SIGIR Forum*, 24(3), 56–61

Pereira, J., & Díaz, O. (2018). A quality analysis of facebook messenger's most popular chatbots. *Proceedings of the ACM Symposium on Applied Computing, Part F1378*, 2144–2150.

Poole, D., Mackworth, A., & Goebel, R. (1997). *Computational intelligence: A logical approach*. New York, NY, USA: Oxford University Press, Inc.

Porter, M. F. (1997). Readings in information retrieval. In K. Sparck Jones & P. Willett (Eds.), (Chap. An Algorithm for Suffix Stripping, pp. 313–

- 316). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Retrieved from <http://dl.acm.org/citation.cfm?id=275537.275705>
- Price, R. (2016). Microsoft deletes racist, genocidal tweets from AI chatbot Tay. Retrieved November 12, 2018, from <http://uk.businessinsider.com/microsoft-deletes-racist-genocidal-tweets-from-ai-chatbot-tay-2016-3?r=US&IR=T>
- Russell, S., & Norvig, P. (2009). *Artificial intelligence: A modern approach* (3rd ed.). Upper Saddle River, NJ, USA: Prentice Hall Press.
- Searle, J. R. (1980). Minds, brains, and programs. *Behavioral and Brain Sciences*, 3(03), 417
- Standard CPP Foundation. (2019). ISO C++. Retrieved March 18, 2019, from <https://isocpp.org/wiki/faq/>
- Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 59(236), 433–460
- Wallace, R. S. (2009). The anatomy of A.L.I.C.E. In *Parsing the turing test: Philosophical and methodological issues in the quest for the thinking computer*.
- Weizenbaum, J. (1966). ELIZA—a Computer Program for the Study of Natural Language Communication Between Man and Machine. *Commun. ACM*, 9(1), 36–45
- Wilensky, R., Chin, D. N., Luria, M., Martin, J., Mayfield, J., & Wu, D. (1988). The Berkeley UNIX Consultant Project. *Comput. Linguist.* 14(4), 35–84.
- Williams, C. (1983). A Brief Introduction to Artificial Intelligence. In *Oceans'83, proceedings* (pp. 94–99). IEEE.
- Wojtusiak, J. (2012). Machine Learning. In N. M. Seel (Ed.), *Encyclopedia of the Sciences of Learning*
- Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd annual meeting on association for computational linguistics* (pp. 189–196). ACL '95

Appendices

A Project Overview

Svetlozar Georgiev

40203970

Initial Project Overview

SOC10101 Honours Project (40 Credits)

Title of Project: Smart Assistant

Overview of Project Content and Milestones

The aim of the project is to create an assistant utilising Artificial Intelligence technology. The main functionality of the software will be:

- The ability to have a conversation with the user. Using Natural Language Processing techniques, the assistant will be able to chat with the user via text messages as if it were a real person.
- The ability to search the Internet for information to answer the user's questions. The information will be summarised, so the user doesn't have to read a whole web page to find what they are looking for.
- The ability to set reminders.
- The ability to communicate with the user via social media direct messaging.

The Main Deliverable(s):

- A report describing the work undertaken and the research carried out to complete the project.
- Backend software sending messages to the user with the use of NLP technologies.
- A database storing information about users. The assistant will be able to learn from past conversations and improve its ability to communicate like a human. Reminders will also be stored in the database.

The Target Audience for the Deliverable(s):

- People who are trying to be more organised.
- People looking for someone to chat with.
- People who would like to practice their written English.
- Researchers looking for an open source project utilising AI and NLP specifically.

The Work to be Undertaken:

- Background research, i.e. reading papers and journals about Artificial Intelligence and Natural Language Processing, studying similar open source projects, testing different AI libraries.
- Choosing an NLP library.
- Managing the project using modern software engineering practices.

Svetlozar Georgiev

40203970

- Developing the backend software. This includes:
 - Creating a piece of software which utilises an NLP library to understand language and respond to questions simulating a human's ability to chat.
 - Using social media APIs so that the assistant can receive messages from the user, analyse them and respond accordingly.
 - Creating a web crawler so that users can request information about anything that can be found online.
 - Using natural language processing to understand this information and summarise it for the user so that they receive only the information they needed.
 - Setting up a web server and a database.
- User testing.
- Bug fixing.
- Evaluation.
- Completing the report.

Additional Information / Knowledge Required:

- AI and NLP and how NLP libraries work.
- Learning how to use social media APIs.
- Expanding knowledge about Python.
- Expanding knowledge on web servers and databases.

Information Sources that Provide a Context for the Project:

- Existing smart assistants, e.g. Cortana, Google Assistant, Siri.
- Existing chatbots, e.g. Cleverbot.
- NLP libraries: spaCy, nltk, Chatterbot, BotMan.
- Jia, J. (2009). CSIEC: A computer assisted English learning chatbot based on textual knowledge and reasoning. *Knowledge-Based Systems*, 22(4), 249–255. <https://doi.org/10.1016/j.knosys.2008.09.001>
- Martinez, A. R. (2010). Natural language processing. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(3), 352–357. <https://doi.org/10.1002/wics.76>
- Young, T., Hazarika, D., Poria, S., & Cambria, E. (2017). Recent Trends in Deep Learning Based Natural Language Processing. *IEEE Computational Intelligence Magazine*, 13(July), 55–75. <https://doi.org/arXiv:1708.02709v6>
- Pereira, J., & Diaz, O. (2018). A quality analysis of facebook messenger's most popular chatbots. *Proceedings of the ACM Symposium on Applied Computing, Part F1378*, 2144–2150. <https://doi.org/10.1145/3167132.3167362>

Svetlozar Georgiev

40203970

- Bhagwat, V. A. (2018). Deep Learning for Chatbots. *Master's Projects*. Retrieved from <http://www.wildml.com/2016/04/deep-learning-for-chatbots-part-1-introduction/>

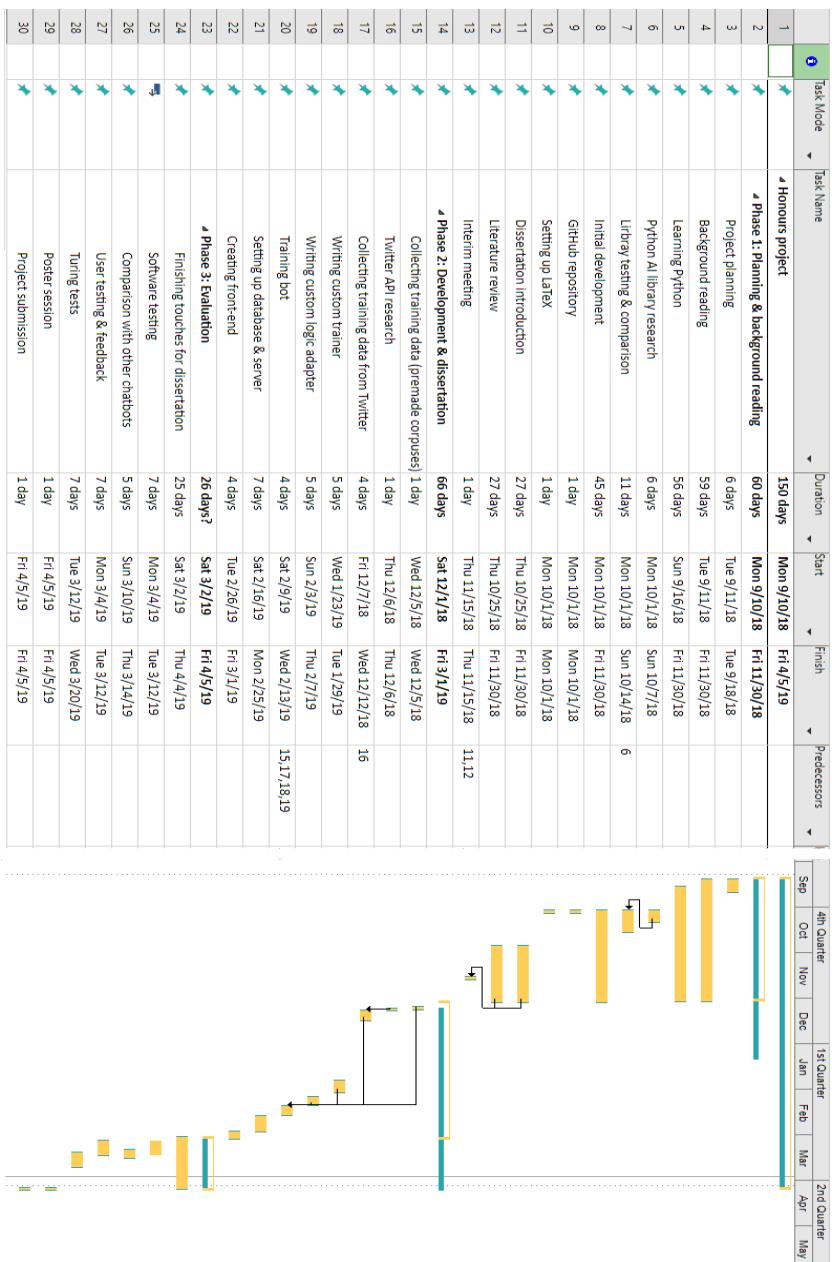
The Importance of the Project:

Even though similar software exists (Google Assistant, Cortana, etc.), it is not open source. Creating an open source project allows other developers to learn from and build up on the existing code. Additionally, similar apps can only answer questions and set reminders. This app will be able to have a conversation with the user like a chat bot. What is more, using social media as the means of communication with the assistant will make the user feel like they are talking to a friend.

The Key Challenge(s) to be Overcome:

- Lack of knowledge about AI and NLP.
- Lack of experience with web technologies.

B Initial Project Plan



C Diary Sheets

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Svetlozar Georgiev

Supervisor: Kevin Sim

Date: 21/09/2018

Last diary date: N/A

Objectives:

- Create a preliminary IPO draft.
- Research LaTeX as advised by supervisor.
- Find papers and journals on AI to prepare for literature review.

Progress:

- Successfully created an initial IPO draft. Some sections have not been fully completed as I have questions for my supervisor.
- Managed to set up MiKTeX and compile the report template provided on Moodle. Also learned how to use Jabref for easier referencing.
- Used Napier library search and Google Scholar to find relevant papers.

Supervisor's Comments:

- Comments on what to change in the IPO.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Svetlozar Georgiev

Supervisor: Kevin Sim

Date: 28/09/2018

Last diary date: 21/09/2018

Objectives:

- Complete IPO based on supervisor feedback.
- Read more papers and journals about AI and NLP.

Progress:

- Read 2 papers about NLP and 1 about creating a chatbot.
- Finished the IPO.

Supervisor's Comments:

- IPO is satisfactory.
- Discussion about papers.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Svetlozar Georgiev

Supervisor: Kevin Sim

Date: 05/10/2018

Last diary date: 28/09/2018

Objectives:

- Continue reading papers and journals
- Investigate available libraries
- Create an example program utilising selected library

Progress:

- Read 2 papers.
- Decided to use the library chatterbot.
- Created 2 example programs. One can be trained using corpora. The second one can find random tweets on twitter and learn from them.

Supervisor's Comments:

- Feedback on example programs.
- Discussion about the selected library.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Svetlozar Georgiev

Supervisor: Kevin Sim

Date: 12/10/2018

Last diary date: 05/10/2018

Objectives:

- Read more papers.
- Start preparing for literature review.
- Improve programs.

Progress:

- Read more papers.
- Found out that a combination of corpora and crawling twitter is probably the best way to train the chatbot.

Supervisor's Comments:

- Discussion about possible training data which can be used.
- Discussion about a paper about a bot which collects data from Twitter.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Svetlozar Georgiev

Supervisor: Kevin Sim

Date: 19/10/2018

Last diary date: 12/10/2018

Objectives:

- Continue training the example programs.
- Continue reading papers.
- Set up the provided Latex template.
- Start writing the literature review.

Progress:

- Changed the custom Twitter trainer, so that it only searches for tweets in English.
- Collected more training data.
- Read 3 more papers.
- Wrote a few paragraphs for the literature review.

Supervisor's Comments:

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Svetlozar Georgiev

Supervisor: Kevin Sim

Date: 26/10/2018

Last diary date: 19/10/2018

Objectives:

- Continue reading papers.
- Continue training the bot.
- Write more paragraphs for the literature review.

Progress:

- Read 2 more papers.
- Collected more data from Twitter. Unfortunately, non-English tweets are still being found.
- Wrote a few paragraphs about Artificial Intelligence for the literature review.

Supervisor's Comments:

- Discussion about data collection and preprocessing.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Svetlozar Georgiev

Supervisor: Kevin Sim

Date: 02/11/2018

Last diary date: 26/10/2018

Objectives:

- Focus on the literature review and get as much done as possible.

Progress:

- Created an initial draft of the introductory A.I. section.
- Created a draft of NLP and Machine Learning sections.

Supervisor's Comments:

- Feedback on the literature review.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Svetlozar Georgiev

Supervisor: Kevin Sim

Date: 09/10/2018

Last diary date: 02/11/2018

Objectives:

- Prepare for the Interim meeting.
- Complete the introduction and chatbot sections of the literature review.

Progress:

- Wrote a draft of the introduction.
- Wrote a draft of the chatbot sections, including history of chatbots.
- Created a Gantt chart.

Supervisor's Comments:

- See interim report (Appendix C)

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Svetlozar Georgiev

Supervisor: Kevin Sim

Date: 21/01/2019

Last diary date: 08/11/2018

Objectives:

Decide what source to use for training data.

Write a web crawler to collect training data.

Train the chatbot with the data and test the results.

Progress:

Decided to use Stack Overflow as data source as suggested by supervisor.

Investigated the Python library BeautifulSoup for web scraping.

Implemented an initial version of the web crawler which can extract text from a Stack Overflow question page.

Had to manually format the data in the yml format required by the chatbot, but managed to train it.

Supervisor's Comments:

Discussion about the lack of progress made during the Christmas break due to broken laptop.

Discussion about possible improvements of the crawler and questions on what topic to scrape.

Discussion about the decision to change the topic of the dissertation as suggested in the Interim Report Meeting in week 9 of semester 1.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Svetlozar Georgiev

Supervisor: Kevin Sim

Date: 28/01/2019

Last diary date: 21/01/2019

Objectives:

Investigate front-end solutions and decide what platform to create the application for.

Improve some problems with the web crawler.

Progress:

Decided to create a website for the front-end as it would be accessible from mobile and desktop devices.

Investigated website development frameworks for Python. Decided to use Flask as it is well documented and provides the necessary features for the development of a small website.

Created a basic prototype of the main page of the website where the user can talk to the chatbot.

Improved the web crawler.

Supervisor's Comments:

Website is a good idea.

Test of the responses of the chatbot trained with Stack Overflow data. Some issues when trying to run the development server on the university WiFi network.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Svetlozar Georgiev

Supervisor: Kevin Sim

Date: 04/02/2019

Last diary date: 28/01/2019

Objectives:

Add basic styling to the website.

Figure out how to connect the front-end and the back-end.

Look into web-hosting options so that demonstrating the project can be done regardless of WiFi network restrictions.

Progress:

Tried to use CSS for the styling of the website, however decided to use a pre-made library due to lack of styling skills. Chose Bootstrap due to previous experience and due to its responsive features.

Managed to send message from the front-end to the back-end and vice versa with JavaScript and JQuery.

Investigated web hosting options and selected PythonAnywhere due to its free options.

Supervisor's Comments:

Discussions about the style and how to make the website more user-friendly.

Supporting mobile devices is a good idea.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Svetlozar Georgiev

Supervisor: Kevin Sim

Date: 11/02/2018

Last diary date: 04/02/2019

Objectives:

Host the website on PythonAnywhere.

Implement simple continuous deployment.

Progress:

Hosted the master branch of the GitHub repository on PythonAnywhere.

Managed to implement a manual deployment method which pulls the changes from GitHub when the master branch is updated.

Supervisor's Comments:

Suggestion to focus on writing the report at this stage.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Svetlozar Georgiev

Supervisor: Kevin Sim

Date: 25/02/2019

Last diary date: 11/02/2019

Objectives:

Implement a feature which allows the user can ask for an alternate response to a previously asked question.

Write some of the Implementation section in the report.

Progress:

Implemented the alternate response feature which required additions to the front-end and the back-end.

Wrote a part of the Implementation section.

Supervisor's Comments:

Discussion about the implemented feature and the progress of the dissertation.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Svetlozar Georgiev

Supervisor: Kevin Sim

Date: 04/03/2018

Last diary date: 25/02/2019

Objectives:

Finish the implementation section of the report.

Add an extra feature which allows the users to rate bot answers to help the bot answer better.

Think about an evaluation strategy and organise and ask people to test the application.

Progress:

Finished most of the Implementation section.

Could not get the new feature to work due to database problems.

Asked people if they are willing to take part in the evaluation, however still waiting for responses.

Supervisor's Comments:

Comments about the implementation section.

Suggestion about how to fix the database problem.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Svetlozar Georgiev

Supervisor: Kevin Sim

Date: 18/03/2019

Last diary date: 04/03/2019

Objectives:

Finish the feature which was not fully implemented last week.

Edit the implementation section.

Prepare a survey for the testers.

Progress:

Managed to fix the problem with the new feature. Tried to clean up some problems and bugs before the testing process.

Created a survey which was sent to all participants who agreed to help.

Started writing the first part of the evaluation section which is a comparison with existing chatbots.

Supervisor's Comments:

Discussion about the evaluation method, the survey and the user testing.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Svetlozar Georgiev

Supervisor: Kevin Sim

Date: 25/03/2019

Last diary date: 18/03/2019

Objectives:

Finish the evaluation section and go through the whole dissertation and make changes where necessary.

Progress:

Finished the evaluation section about adding the results of the survey.

Started editing the dissertation.

Supervisor's Comments:

Feedback on dissertation structure and contents.

D Second Formal Review Output

SOC10101 Honours Project (40 Credits)

Week 9 Report

Student Name: Svetlozar Georgiev

Supervisor: Kevin Sim

Second Marker: Neil Urquhart

Date of Meeting: 14/11/2018

Can the student provide evidence of attending supervision meetings by means of project diary sheets or other equivalent mechanism? yes no*

If not, please comment on any reasons presented

Diary Sheets Supplied - Good Attendance

Please comment on the progress made so far

Excellent 1st review.

Researched libraries + tested chatbox within tool by creating a test proj. Experimented with Turtle-Click + decided not to use it within this proj.

Is the progress satisfactory? yes no*

Can the student articulate their aims and objectives? yes no*

If yes then please comment on them, otherwise write down your suggestions.

* Please circle one answer; if no is circled then this must be amplified in the space provided

Does the student have a plan of work? yes no*

If yes then please comment on that plan otherwise write down your suggestions.

Does the student know how they are going to evaluate their work? yes no*

If yes then please comment otherwise write down your suggestions.

Think carefully about compensators with other
chatbots. You need a framework for
compensation - e.g. compare the ability to
answer/respond to specific questions.

Any other recommendations as to the future direction of the project

Neil suggested implementing, using a
case study such as answering student's
questions to make it easier to evaluate

Signatures: Supervisor cl S Second Marker Neil Wright

Student Svetlozar

The student should submit a copy of this form to Moodle immediately after the review meeting; A copy should also appear as an appendix in the final dissertation.

* Please circle one answer; if no is circled then this must be amplified in the space provided

E Results from the First Testing Session

Table 4: Results from the first testing session.

Feedback	Changes made
Links are not identifiable easily.	The colour of all links was changed to stand out from regular text. The style was also changed so that they are underlined.
The “help” link is not visible.	The style for this link was also changed. Additionally, another help button was added to a menu at the top of the webpage.
It would be more user-friendly to add an “alternate response” button.	An alternate response button was added to the bottom of each bot message.
When the user gives negative feedback, they should be presented with an alternate response button.	An alternate response button was added to the template for the “no” feedback button.
The webpage fills the screen and answers which are too long look like a wall of text. It is difficult to read.	The initial width of the chat box was decreased.
The help message should describe the features of the application more clearly.	The help message was edited.
There should be a description of the website for users who accidentally found it.	Another webpage was added which describes the purpose of the website.

F Survey Questions

Table 5: The questions each tester was asked in the survey and their possible answers.

ID	Question	Possible Answers
1	What is your overall programming knowledge?	Beginner/Intermediate/Advanced
2	The user interface is easy to understand and use.	Scale from “Strongly Agree” to “Strongly Disagree”
3	The user interface is visually pleasing.	Scale from “Strongly Agree” to “Strongly Disagree”
4	I was able to understand how to use all the features of the application, i.e. asking for a regular response , asking for an alternate response , and providing feedback for a bot response.	Scale from “Strongly Agree” to “Strongly Disagree”
5	I was able to use the alternate response feature to receive an alternate answer to a question I asked.	Scale from “Strongly Agree” to “Strongly Disagree”.
6	I was able to use the feedback feature to give feedback on an answer the chatbot gave me.	Scale from “Strongly Agree” to “Strongly Disagree”
7	Appropriate error messages are shown (e.g. when there is no user input).	Scale from “Strongly Agree” to “Strongly Disagree”
8	The chatbot answered my question(s) quickly.	Scale from “Strongly Agree” to “Strongly Disagree”

Continued on next page

Table 5 – continued from previous page

ID	Question	Possible Answers
9	The chatbot response(s) were coherent and answered my question(s) accurately.	Scale from “Strongly Agree” to “Strongly Disagree”
10	I am overall satisfied with the functionality of the application.	Scale from “Strongly Agree” to “Strongly Disagree”
11	Any additional feedback?	Text input field where the user can write their additional feedback.

G Web Crawling Program

Listing 16: The source code of the second web crawling program created for data collection from the ISO C++ website.

```

1 import requests
2 from ruamel.yaml import YAML
3 from bs4 import BeautifulSoup as bs4
4 import os
5 import threading
6
7 data = []
8 DATA_DIR = os.path.join(os.getcwd(), 'data')
9 FILE_EXT = '.yaml'
10
11
12 def write_to_file():
13     formatted_data = dict(categories=['faq', 'cpp'],
14                           conversations=data)
15
16     if not os.path.exists(DATA_DIR):
17         os.makedirs(DATA_DIR)
18
19     yaml = YAML()
20     yaml.default_flow_style = False
21
22     out_file = os.path.join(DATA_DIR, str(threading.
23                            get_ident()) + FILE_EXT)
24
25
26
27 def crawl_pages(url):
28     page_src = requests.get(url).text
29     soup = bs4(page_src, 'lxml')
30
31     for a in soup.findAll('a'):
32         href = a['href']
33         if href.startswith('/wiki'):
34             a['href'] = a['href'].replace(href, 'https
35                                         ://isocpp.org' + href)
```

```
36     result = soup.findall(['h3', 'p', 'pre', 'ul', 'ol',
37                             , 'li', 'code'])
38     question = None
39     answer = []
40     found_question = False
41
42     for r in result:
43
44         if found_question and question is not None:
45             str_r = str(r)
46
47             if str_r.startswith('<p') or str_r.
48                 startswith('<pre') or str_r.startswith(
49                     '<ul') or str_r.startswith('<ol'):
50                 answer.append(str(r))
51
52             elif str(r).startswith('<h3'):
53                 j_answer = '\n'.join(answer)
54                 entry = [question, j_answer]
55                 data.append(entry)
56                 answer.clear()
57                 found_question = False
58                 question = None
59
60             if str(r).startswith('<h3'):
61                 found_question = True
62                 question = str(r.text)
63
64
65 if __name__ == "__main__":
66     urls = ['https://isocpp.org/wiki/faq/big-picture', '
67             https://isocpp.org/wiki/faq/newbie', ]
68     for url in urls:
69         crawl_pages(url)
```
