Using Natural Language Processing and Machine Learning to address the Language Learning Plateau

by

Khalil Hervin Russell

URN: 6444817

Supervisor: Gregory Chockler

A final year project submitted in partial fulfilment of the requirements for the award of

BACHELOR OF SCIENCE IN COMPUTER SCIENCE

May 2021

Declaration of Originality

I confirm that the submitted work is my own work and that I have clearly identified and fully acknowledged all material that is entitled to be attributed to others (whether published or unpublished) using the referencing system set out in the programme handbook. I agree that the University may submit my work to means of checking this, such as the plagiarism detection service Turnitin®UK. I confirm that I understand that assessed work that has been shown to have been plagiarised will be penalised.

Acknowledgements

I would like to thank my family and friends for their support and encouragement during my time at university. Especially, my mother, father and Aunt Michelle who have read through this entire dissertation. I would also like to say thank you to Sparkle Russell for her invaluable knowledge and my supervisor, Professor Gregory Chockler, for his guidance throughout this project.

Abstract

Machine learning is the development of computer systems that use algorithms and statistical models to be able to automatically learn and improve from their own experiences without being explicitly programmed or taught. Drawing from inferences and patterns from the data that they are fed these systems are capable of achieving human like performance on a specific task. Natural language processing (NLP) is a subfield of Artificial Intelligence that deals with the interaction between computers and humans. This field gives machines the ability to read, understand and derive meaning from human languages. A chatbot is a computer program created to simulate conversation with a user through text, speech or both.

This project will explore using all of these elements in combination to produce a system, that through analysing the conversation between a user and a chatbot in the form of messages, can determine a user's proficiency in Mandarin Chinese. Based on this analysis the system will also make suggestions on areas that the users should concentrate on to improve their proficiency.

Table of Contents

1 Introduction....9
1.1 Problem Background....9
1.2 Project Description....10
1.3 Aims and Objectives....10
1.4 Planning....10
1.5 Structure of Report....11
1.6 Summary....11
2 Literature Review....12
2.1 Chatbots in Foreign Language learning for Chinese....12
2.1.1 Mondly Chatbot....12
2.1.2 Eggbun Education chatbot....13
2.1.3 Conclusion....14
2.2 Chatbot Types....14
2.2.1 Classification by knowledge domain....15
2.2.2 Classification by service....15
2.2.3 Classification by goals....15
2.2.4 Classification by amount of human-aid....16
2.2.5 classification by input process and response generation method....16
2.2.6 Conclusion....16
2.3 Types of Chinese Proficiency Exams....16
2.3.1 汉语水平考试 Hànyǔ Shuǐpíng Kǎoshì (HSK)....17
2.3.2 Test Of Chinese as a Foreign Language (TOCFL)....17
2.3.3 Conclusion....17
2.4 Related Research....18
2.4.1 TransferTransfo: A Transfer Learning Approach for Neural Network Based Conversational Agents....18
2.4.2 A Large-Scale Chinese Short-Text Conversation Dataset....19
2.4.3 Retrieve and Refine:Improved Sequence Generation Models For Dialogue....21
2.4.4 Conclusion....21
2.5 Summary....21
3 Systems Requirements and Specification....22
3.1 Project Scope....22
3.2 Functional Requirements....22
3.2.1 Must Requirements....22
3.2.2 Should Requirements....24
3.3 Non-functional Requirements....25
3.4 Software Development Lifecycle Methodologies....25
3.5 Summary....26
4 System Design....27
4.1 Machine Learning Model Architectures....27
4.1.1 Chatbot Model....27
4.1.2 Classification Model....28
4.2 Use Case Diagram....29
4.3 Activity Diagrams....30
4.4 System Architecture Design....32
4.5 Security Design....34
4.6 Database Design....36
4.6.1 Entity Relationship Diagrams (ERDs)....36
4.6.2 Database Management Systems....37

4.7 Docker Image Design....40
4.7.1 User Interface Service....40
4.7.2 Chatbot Service....41
4.7.3 Grammar Extractor Service....41
4.7.4 Classifier Service....41
4.7.5 User Authentication Service....42
4.7.6 CoreNLP Service....42
4.7.7 Database Service....42
4.8 API Design....42
4.8.1 Chatbot Service API....42
4.8.2 Grammar Extractor Service API....42
4.8.3 Classifier Service API....42
4.8.4 User Authentication Service API....43
4.9 User Interface....43
4.9.1 CSS Frameworks....43
4.9.2 User Interface Design....44
4.10 Summary....46
5 Implementation....47
5.1 Codebase Structure Overview....47
5.2 Docker....48
5.2.1 Example Docker File....48
5.2.2 Docker Compose Files....49
5.3 Chatbot Service....51
5.4 Classifier service....53
5.5 Grammar Extractor service....56
5.6 User Interface Service....59
5.6.1 Context....59
5.6.2 Higher-Order Component (HOC)....60
5.7 User Authentication Service....62
5.8 Summary....65
6 Testing....65
6.1 Unit Testing....65
6.1.1 Unit Testing Environment Docker Compose....65
6.1.2 Endpoint Unit Testing....66
6.1.3 Chatbot Service Additional Unit Testing....68
6.1.4 Classifier Service Additional Testing....69
6.1.5 Grammar Extractor Service Additional Unit Testing....71
6.1.6 Test coverage....72
6.2 Integration Testing....74
6.3 System & Validation Testing....75
6.4 Summary....84
7 Evaluation & Conclusion....85
7.1 Evaluation of project against objectives....85
7.2 Evaluation of planning....87
7.3 Evaluation of the system....87
7.4 Future Work....88
7.5 Conclusion....89
8 Statement of Ethics....90
8.1 Legal Acts....90
8.2 Social Considerations....91
9 Bibliography....92

10 Appendices....96
10.1 Append A – Gantt Chart for project....96
10.2 Appendix – B SAGE Form....99
10.3 Appendix C – Application access through web clients....103
10.4 Appendix – D GPT3 Chatbot Example....104

Table of Figures

Figure 1: Mondly Chatbot....12
Figure 2: Eggbun Education Chatbot Reply Type 1....13
Figure 3: Eggbun Education Chatbot Reply Type 2....14
Figure 4: Chatbot Automatic Metric Evaluation [23]....19
Figure 5: Chatbot Human Evaluation[23]....20
Figure 6: GPT Model Architecture [29]....27
Figure 7: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention[30]....28
Figure 8: MVP Use Case Diagram....29
Figure 9: Use Case Diagram with FS1 Requirement....30
Figure 10: Activity Digram showing a user registering a new account....31
Figure 11: Activity Diagram showing user logging into the system....32
Figure 12: MVP Prototype Micro-service System Architecture....33
Figure 13: Prototype Micro-service System Architecture with FS1 Requirement....34
Figure 14: Entity Relationship Diagram for MVP system....36
Figure 15: Entity Relationship Diagram for prototype System with FS1 Requirement....37
Figure 16: Application Logo....44
Figure 17: UI Mock-up of Registration Page....45
Figure 18: UI Mock-up of Login Page....45
Figure 19: UI Mock-up of Chatbot Page....46
Figure 20: Codebase Main Directory....47
Figure 21: Codebase Service Directory....47
Figure 22: User Authentication Service Directory....47
Figure 23: User Interface Service Src Directory....47
Figure 24: User Authentication Service Docker File....48
Figure 25: Docker Compose Classifier Service Definition....49
Figure 26: Docker Compose Network Definition....49
Figure 27: Docker Compose Classifier Service Development Configuration....50
Figure 28: Docker Compose Database Service Definition....50
Figure 29: Chatbot Chat Function Loop and Conditional Fix....51
Figure 30: Chatbot Sentence Similarity Conditional....52
Figure 31: Classification Model Training loop....54
Figure 32: Validation Accuracy of Top 5 Trial Models....55
Figure 33: coreNLP Service Example Response....57
Figure 34: Grammar Extractor Example POS Sentence....57
Figure 35: Grammar Dictionary One Pattern....57
Figure 36: Grammar Dictionary Multiple Patterns....57
Figure 37: Grammar Extractor Parse Sentence Function....58
Figure 38: Grammar Extractor Extract Grammar Function....58

Figure 39: User Interface Logout Function....59
Figure 40: User Interface Check Auth Function....59
Figure 41: User Interface Auth Context Provider....60
Figure 42: Auth Provider Wrapping Application....60
Figure 43: Auth Route Use State and Use Effect React Hooks....61
Figure 44: Auth Route Render Conditionals....61
Figure 45: Classify Endpoint Function....62
Figure 46: Chat Web-socket Endpoint Function....63
Figure 47: Web-socket Proxy Functions....64
Figure 48: Docker Compose Testing Configurations....65
Figure 49: Classifier Service Endpoint Unit Test....66
Figure 50: Pytest Client Fixture....67
Figure 51: Grammar Extractor Crud Function Unit Test....68
Figure 52: Chatbot Similarity Functionality Unit Tests....69
Figure 53: WordLevelClassifier Class Unit Tests....69
Figure 54: Classifcation Model Classification Report....70
Figure 55: Grammar Regex Patterns Test Function....71
Figure 56: Grammar Extractor Parse Sentence and Extract Grammar Test Functions....71
Figure 57: Chatbot Service Unit Test Coverage....72
Figure 58: Classifier Service Unit Test Coverage....72
Figure 59: Grammar Extractor Service Unit Test Coverage....73
Figure 60: User Authentication Service Unit Test Coverage....73
Figure 61: Grammar Extractor Integration Test....74
Figure 62: Application Accessed Via Chrome....102
Figure 63: Application Accessed Via Firefox....102

Table of Tables

Table 1: Report Structure....11
Table 2: Must Requirements....24
Table 3: Should Requirements....25
Table 4: Non-functional Requirements....25
Table 5: Evaluation of DBMS....40
Table 6: Chatbot Service API Endpoints....42
Table 7: Grammar Extractor Service API Endpoints....42
Table 8: Classifier Service API Endpoints....43
Table 9: User Authentication Service API Endpoints....43
Table 10: Bootstrap and Material Design Evaluation....44
Table 11: Average Model Validation Accuracies on the Different Proficiency Test Vocabulary Datasets 55

1 Introduction

1.1 Problem Background

Over the last two decades the number of people deciding to study Mandarin Chinese as a second language has increased dramatically, with there being an estimated 100 million people studying Mandarin around the world in 2017; an increase from 30 million in 2004 [1]. In the UK, this trend is especially apparent, with the government in 2015 pledging a £10 million funding boost to ensure that 5,000 more students will be learning Mandarin in primary and secondary schools by 2020 [2]. This change was probably spurred by the increase in China's economic status. Since opening its boarders for trade and investment and introducing economic reform in 1979, China has experienced rapid economic growth [3] becoming the world's 2nd largest economy in 2010 [4]. However, Mandarin is considered to be one of the hardest languages for native English speakers to learn [5]. This is mainly due to its character-based writing system that completely differs to the Latin alphabet and, the use of a four tone tonal system (six for the more advanced speakers) to convey meaning rather than emotion like in English. For example, the words 水 and 税 are both pronounced as Shui however one is pronounced with the third tone Shuǐ and the other with the fourth tone Shuì. They also have completely different meanings, 水 means water and 税 means tax.

There are a number of different language learning resources available to learners to aid them in their language learning journey which include: textbooks, language tutors both online and in person, multiple online courses, podcasts, blogs, mobile apps and flashcard applications. Many of these online courses and mobile apps utilise machine learning and chatbots to enhance their user's experience with some popular examples of this being Mondly, Eggbun Education and Duolingo. Many of these language learning applications have seen a surge in usage due to the COVID-19 pandemic [6].

However, despite having all these resources available to learners there is still a common issue that most language learners will encounter, known as the the language learning plateau. This occurs when the language learner's progress begins to slow down and are not learning as quickly as they previously were. In the early stages of the language learning processes they are able to make quick and meaningful progress, but as they acquire more knowledge, naturally their rate of progress slows down. For this reason, a language learning plateau frequently occurs when learners reach an intermediate level of language proficiency [7]. The purpose of this final year project is to produce a technological solution that tries to address this issue.

There is also some individual motivation for this project as a student of Mandarin Chinese for several years. After passing the HSK 4 exam (汉语水平考试四级), a test of Chinese proficiency, and attaining a fairly highly level of proficiency, the language learning plateau is still a challenge being personally overcome.

1.2 Project Description

This project documents the process of using machine learning and natural language processing techniques to produce a prototype of a system that can aid a user in improving their level of Chinese proficiency. Machine learning will be used in two main aspects of the project: first, to create a chatbot that can converse with users in Chinese and second, to produce a classification model that will be able to assign Chinese words to a level of Chinese proficiency. Natural language processing will be used to create a pipeline that can extract Chinese grammar patterns from Chinese sentences, and also assign these grammar patterns a level of Chinese proficiency. Next, an automated feedback system will have to be produced that can take the user proficiency information as an input, and produce a set of suggestion on how to improve proficiency as an output. Finally, these different aspects will be combined together and presented to the user as a web application. Using this web application the user will converse with the chatbot and the system will use their inputs during the conversation to assess their current level of proficiency before returning a set of suggestions on how they can improve. This prototype will be developed using the cloud-native design pattern so is readily deployable in a cloud native environment.

1.3 Aims and Objectives

The main aim of this project is to produce a prototype system, that through analysing the conversation between a user and a chatbot in the form of messages, can determine a user's proficiency in Mandarin Chinese. Based on this analysis the system will also make suggestions on areas that the users should concentrate on to improve their proficiency. A multitude of different machine learning and natural language processing (NLP) techniques will be explored and leveraged in the creation of this solution. The final solution will be assessed against the initial project requirements and constraints to decide it's level of success.

To achieve the overall goal the following objectives should be completed (not necessarily in the stated order):

- Explore some of the existing implementations of Chinese chatbots used in language learning
- Research some of the existing proficiency assessments and how they are assessed
- Decide on a set of functional and non-functional requirements for the project
- Produce designs for the different components of the system based on the defined requirements
- Develop a Chinese Chatbot
- Develop a machine learning classification model that can determine the proficiency level of a Chinese word
- Develop a NLP pipeline that can extract grammar patterns from Chinese text and assign them a proficiency level
- Develop a proficiency improvement automated feedback system
- Develop a web application combining these different components
- Conduct a robust set of tests to evaluate the functionality of the system
- Evaluate the success of the project and propose possible improvements

1.4 Planning

A detailed Gantt chart describing the project plan can be found in Appendix A. This chart sets out the length of time allocated for each of the deliverables of project but also takes into account the workloads associated with the other modules being taken this year. For example time is allocated to prepare for the January examination period during the first semester.

1.5 Structure of Report

The report will follow the structure presented below

Chapter	Description
Introduction	In this section the problem that the project will be attempting to solve will be introduced. The aims and objectives of the project are highlighted, and a work plan is set out. The structure of the report is also clarified at the end of this section.
Literature Review	This section describes the investigation that was carried out on existing implementations of chatbots in foreign language learning for Chinese and the different types of chatbots. The most popular Chinese proficiency assessments are examined before finally, a critical review of relevant research papers are carried out.
System Requirements and Specification	In this section the scope of the project is discussed before a set of functional and non-functional requirements are selected. Different Software Development Lifecycle models are also critiqued before an appropriate model choice is made for the project
System Design	This section outlines the designs created for the system before its implementation. The system architecture, security and database designs are examples of some of the designs that will be detailed with justifications for each design choice shown.
Implementation	In this section the implementation process of the system is documented. The different technologies and methods applied and the accompanying challenges faced will also be detailed
Testing	In this section the testing process completed on the system is set out.
Evaluation and Conclusion	This section evaluates the implemented system against the requirements previously defined and uses this to assess the level of success of the project. Conclusions are drawn from this assessment and future development of this system is also discussed
Statement of Ethics	This section reviews the statement of ethics for this project and will consist of the social, legal, ethical and professional considerations made

Table 1: Report Structure

1.6 Summary

In this section the problem area trying to be addressed by the project was introduced and a high-level description of the project was provided. Along with the project aims and objectives, the planning of the project was presented and the structure of the report was outlined

2 Literature Review

This section will discuss the existing implementations of chatbots in foreign language learning; the different proficiency exams available for Chinese; research into the different types of chatbot followed by a critical analysis of existing research papers relating to the problem area of the project.

2.1 Chatbots in Foreign Language learning for Chinese

Chatbots are becoming increasing popular in the Foreign Language Learning (FLL) field. This popularity stems from the different advantages they have compared to face interactions with other humans when practising a language. Firstly, a number of people experience anxiety when practising a foreign language with a human partner [8], but feel more comfortable talking to a chatbot as it removes the fear of being judged. Also, a chatbot can go over the same material repeatedly without getting bored or losing patience, and can be accessed anytime and anywhere [9]. In this section two examples of existing chatbots used in foreign language learning for Chinese will be reviewed.

2.1.1 Mondly Chatbot

Figure 1: Mondly Chatbot

Mondly is an educational technology company founded in 2013 that develops a language learning application available as a web and mobile application. The Mondly application offers free and paid language courses for 33 languages including Chinese [10]. The Mondly Chatbot was the first chatbot

with speech recognition on iTunes and Google Play and converses with users based around a specific topic i.e. restaurants. The chatbot actively asks users questions and replies to user responses in Chinese, with the audio for each response also being played in Chinese. On receiving a message from the chatbot a list of possible replies to the message are generated, with each reply also having their audio played for the user. Users can then pick one of those generated replies or type their own and can reply using either Chinese characters or Pinyin (Romanization of the Chinese characters based on their pronunciation).

The chatbot conversation also supports some customizability. Pressing the symbol in the top right corner of the chatbox switches between displaying messages using Chinese characters or Pinyin. Additionally, the translations toggle in the bottom left removes the English translation from all messages and generated replies. These are beneficial features as they allow the user to have some control over the difficulty of the conversation. However, this is all the control the user has as the chatbot is in complete control of the flow of conversation. Any messages that deviate from this flow are also not address and the chatbot simply continues as if the response given was as expected even if it was completely different. It is possible that there are more free flowing chatbot conversation available in later lessons or for the premium paying users, but the personnel, time and financial resources to explore this possibility were not available.

2.1.2 Eggbun Education chatbot

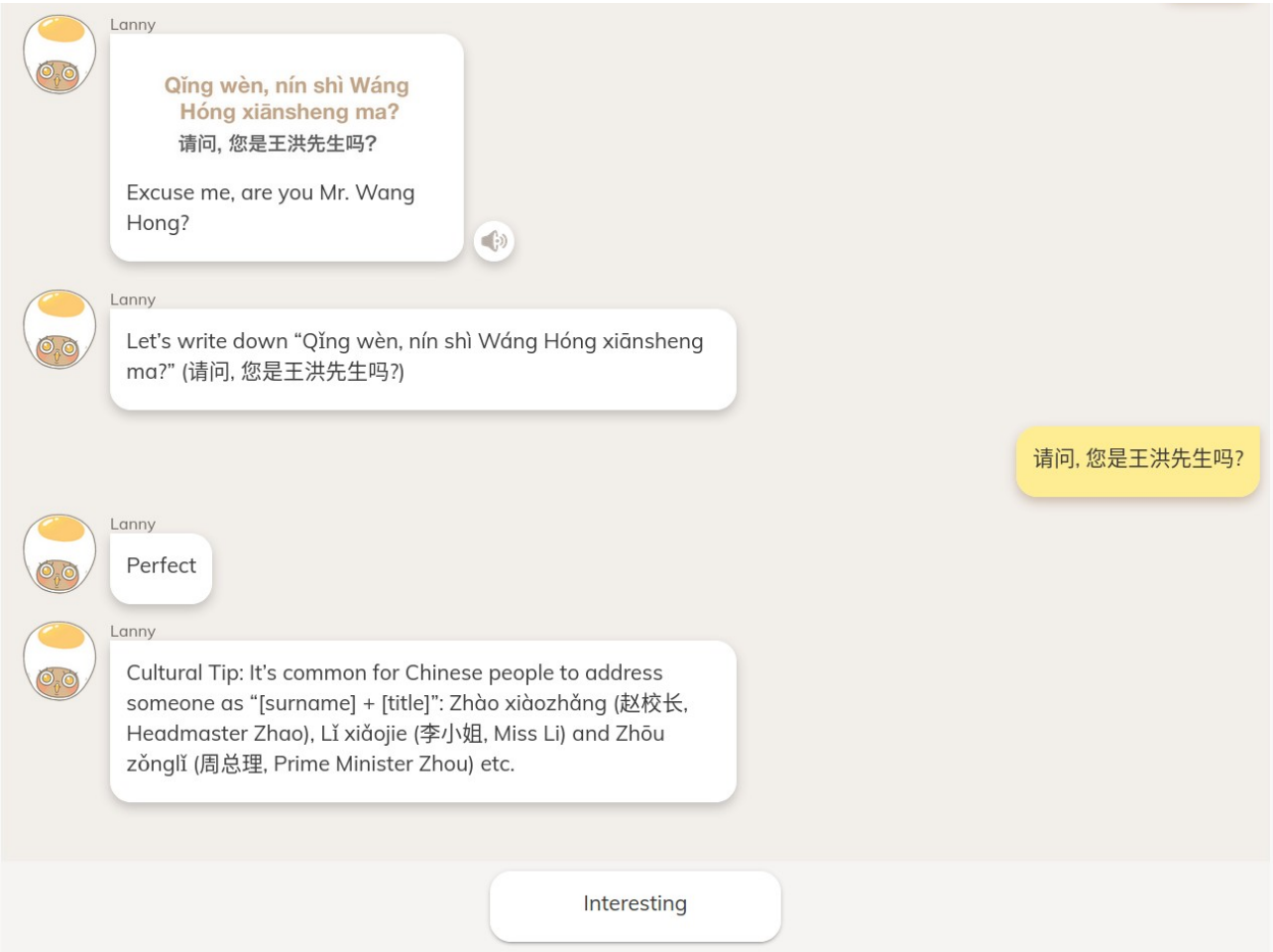


Figure 2: Eggbun Education Chatbot Reply Type 1

Figure 3: Eggbun Education Chatbot Reply Type 2

Eggbun Education initially created their language learning application, available as a web and mobile application, to teach Korean. However, they have recently expanded its capabilities to also include Japanese and Chinese. The Eggbun Education chatbot, named Lanny, takes users through a lesson in English where users are either given set responses (Figure 2) or are prompted to type certain Chinese phrases taught throughout the lesson (Figure 3). Just like the Mondly chatbot, the typed replies can be written using either Chinese characters or Pinyin. Again, the user has no control over the flow of conversation due to the use of set responses, and the chatbot promoting users to enter typed responses until it receives the response it expects. Resembling Mondly, there are premium lessons where the chatbot functionality may improve but the personnel, time and financial constraints still apply.

2.1.3 Conclusion

After reviewing these two examples of chatbots used in foreign language learning for Chinese, it is clear that, although both have several beneficial features, they lack the ability to have free flowing, spontaneous conversation with users. This is an area the project aims to address to provide users with a more immersive experience when studying Chinese.

2.2 Chatbot Types

There are multiple ways to classify chatbots based on: the knowledge domain, the service provided, the goals, the amount of human-aid and the input processing and response generation method [11].

2.2.1 Classification by knowledge domain

Classification by knowledge domain bases its classification on the knowledge that can be accessed by the chatbot or the amount of data they are trained on. When classify by knowledge domain a chatbot can either be open domain and can talk about general topics, or closed domain meaning it is focused on a particular topic and may fail to respond to questions outside of that [11]. For this project, as the aim is to use the chatbot conversation as a form of proficiency assessment, an open domain chatbot would be better suited. This is because a closed domain chatbot would only be able to converse on specific topic, meaning the conversation generated would not a be full representation of the user's full proficiency but rather just their proficiency on that topic. An open domain chatbot would produce a more balanced conversation, making it a better source to measure a user's proficiency.

2.2.2 Classification by service

Classification by service provided looks at the sentimental proximity between the chatbot and the user as well as the amount of intimate interaction that takes place. This makes it also dependent upon the task that the chatbot is performing. The classification classes for this form of classification are interpersonal, intrapersonal and inter-agent. An interpersonal chatbot lies within the social or personal proximity zones and provide services such as booking flights or answering FAQs. These chatbots are focused on getting information to the user and do not need to be friendly or have any sort of personality. An intrapersonal chatbot lies within the personal proximity zone with a focus of being a companion to the user and helping them with more personal tasks. These tasks could include managing a calendar or storing personal opinions. Lastly, inter-agent chatbots lie outside the proximity zones in the IOT domain with a focus on communication with another bot to complete a task [12].

Clearly an inter-agent chatbot does not fit this project as the communication will be between a human and chatbot, not between two bots. A interpersonal chatbot would also be a poor fit as there is no specific information that needs to be passed to the user. An intrapersonal chatbot would be the most suitable for this project due their focus of acting as a companion, this will be important when trying to create conversation with the user.

2.2.3 Classification by goals

Classification by goals is based on the primary goal or aim of the chatbot and classifies chatbots as either informative, chat-based or conversational and task-based. The goal of informative chatbots is to provide the user with information that is stored beforehand or is available from a fixed source for instance, a database. The goal of chat-based or conversational chatbots is to talk to the user by responding correctly to the sentence they have been given. Finally, the goal of task-based chatbots is to perform a specific task, for example booking a flight or a making restaurant reservation [12]. Considering this type of chatbot classification, it is clear that the type of chatbot necessary for this project would be the chat-based or conversational chatbot. An informative chatbot would not make sense for this project's use case as there is no explicit information for the chatbot to provide. A task-based chatbot is also ill-suited as, like the interpersonal chatbot, there is no specific task that the chatbot needs to complete inside this system other than facilitating conversation.

2.2.4 Classification by amount of human-aid

Classification by the amount of human-aid considers a chatbot to be human-aided if at least one of its components uses human computation. This normally takes the form of an employee embodying their intelligence in the chatbot logic, helping to compensate for the gaps of purely automated chatbots [13]. A human-aided chatbot would not be suitable for this project due there not being an employee available to perform the human computations on a full-time basis. There also the disadvantage of human computation being slower than computer computation which introduces a problem of scalability for multiple user interactions.

2.2.5 classification by input process and response generation method

The final classification type being considered is classification by input process and response generation method which has the classification classes of rule-based, retrieval-based, generative and hybrid. For rule-based chatbots, the type that most of the first chatbots were, the responses of the chatbots are based on a fixed set of hand-coded, predefined rules, and pattern matching is used to produce the correct response. Retrieval-based chatbots are similar to rule-based chatbots but provide more flexibility. Instead of pattern matching the exact input to the exact response, the intent of the response is extracted and pattern matching is performed on the intent to retrieve and return the correct response. Generative chatbots use machine learning algorithms and deep learning techniques to take current and previous user messages and to build a unique response. Finally, hybrid chatbots use a combination of the above classifications to produce a response, for example a system that uses a flow chart for managing the direction of conversation but provides responses that are generated .

For the purpose of this project the rule-based chatbot is unsuitable for multiple reasons [11], [12]. Firstly, a similar problem would be faced as with a closed domain chatbot in the sense that the conversation generated would not give a full reflection of a user’s proficiency. Secondly, producing a set of predefined rules for conversation that would produce a suitable conversation for assessment would simply not be feasible to implement given the time constraints. Retrieval-based chatbots would also be unsuitable, as although they would generally be a better fit than the rule-based chatbot, a similar issue of creating a extensive set of intents within the time constraints is still present. Both the generative and Hybrid chatbots would be the best choices of this project. The generative chatbot will be implement and could be extended to a hybrid if the time constraints of the project permit this.

2.2.6 Conclusion

In conclusion, combining all the different classification types of chatbots, the ideal chatbot that will need to be produced for this prototype system will be an: open domain, intrapersonal, conversational, generative or hybrid chatbot.

2.3 Types of Chinese Proficiency Exams

There are number of different Chinese proficiency exams available to test the four different areas of language: reading, writing, listening and speaking. To ensure the validity of the proficiency assessment conducted by the system, the assessment methods will be as closely aligned to one of these proficiency exams as possible. For the purpose of this project the focus will be on the proficiency exams that look at reading and writing, as these are the two areas of proficiency that the system will be able to assess through a users conversation with the chatbot. Most of these also incorporate listening and of the exams

available the most popular and internationally recognisable are the 汉语水平考试 Hànyǔ Shuǐpíng Kǎoshì (HSK) and the Test Of Chinese as a Foreign Language (TOCFL).

2.3.1 汉语水平考试 Hànyǔ Shuǐpíng Kǎoshì (HSK)

The HSK exam is an international standardized test of Chinese language proficiency for non-native speakers i.e., foreign students administrated by the Ministry of Education of the People's Republic of China. The test aims to be a certificate of language proficiency for higher educational and professional purposes. As currently constructed, it consists of 6 levels. HSK Level 1 (HSK 一级, HSK Level I) equates to the A1 Level of the Common European Framework of Reference (CEF) and requires students to have mastered 150 commonly used words and basic grammar patterns. Level 2 (二级, Level II) equates to the A2 level of the CEF and requires students to have mastered 300 commonly used words and their related grammar patterns. Level 3 (三级, Level III) equates to the B1 Level of the CEF and requires students to have mastered 600 commonly used words and their related grammar patterns. Level 4 (四级, Level IV) equates to the B2 Level of the CEF and requires students to have mastered 1200 commonly used words and their related grammar patterns. Level 5 (五级, Level V) equates to the C1 Level of the CEF and requires students to have mastered 2500 commonly used words and their related grammar patterns. Finally, level 6 (六级, Level VI) equates to the C2 Level of the CEF and requires students to have mastered 5000 or more commonly used words and their related grammar patterns [14].

2.3.2 Test Of Chinese as a Foreign Language (TOCFL)

The Test Of Chinese as a Foreign Language (TOCFL) is Taiwan's national standardized test of Chinese (Taiwanese Mandarin) language proficiency of non-native speakers, such as foreigners and overseas Chinese. The TOCFL test is administrated by the Ministry of Education through the Steering Committee for the Test of Proficiency-Huayu (SC-TOP). Like the HSK the test provides a certificate of language proficiency for higher educational and professional purposes. Unlike the HSK the TOFCL has 8 levels, novice levels 1 & 2, then level 1 to 6 that equate to the CEF. Novice 1 requires students to have mastered 150 commonly used words while Novice 2 requires students to have mastered 300 commonly used words. Level 1 equates to the A1 Level of the CEF and requires students to have mastered 500 commonly used words. Level 2 equates to the A2 Level of the CEF and requires students to have mastered 1000 commonly used words. Level 3 equates to the B1 Level of the CEF and requires students to have mastered 2500 commonly used words. Level 4 equates to the B2 Level of the CEF and requires students to have mastered 5000 commonly used words. Level 5 equates to the C1 Level of the CEF and requires students to have mastered 8000 commonly used words. Finally, Level 6 equates to the C2 Level of the CEF and also requires students to have mastered 8000 commonly used words [15].

2.3.3 Conclusion

Both of these proficiency tests have a clear structure and vocabulary lists that can be used as a source of training data for the proficiency classification model. Although the TOCFL is for Taiwanese Mandarin that uses traditional Chinese characters it would be a simple task to convert them to simplified Chinese characters, which is the more widely used character set . The HSK is far more popular than the TOCFL as it is administrated by the Ministry of Education of the People's Republic of China, however TOCFL does provide a larger vocabulary list which could improve the accuracy of the proficiency classification model. For this project, the performance of the proficiency classification model trained on the different

vocabulary lists will be compared and the highest performing one will be included in the final prototype system.

2.4 Related Research

2.4.1 TransferTransfo: A Transfer Learning Approach for Neural Network Based Conversational Agents

In 2018 Wolf et al. entered and won the Conversational Intelligence Challenge 2 (ConvAI2), a dialogue competition at NeurIPS 2018 [16]. The aim of the competition was to produce a chatbot using the crowd-sourced Persona-Chat dialogue dataset that addressed the weakness of chit-chat models at the time. Those weaknesses were: a lack of a consistent personality, a lack of an explicit long-term memory and the tendency to produce non-specific answers like “I don’t know” [17]. To win this competition Wolf et al. introduced a new method which they called “TransferTransfo” which is a combination of using a transfer learning based training scheme with a high-capacity transformer model. This method resulted in the following metrics that made their method the clear winner: a perplexity score of 16.28 which was a 45% absolute improvement, a Hits@1 score of 80.7 which was a 46% absolute improvement and a F1 score of 19.5 which was a 20% absolute improvement [18].

The generative model used in the “TransferTransfo” method was a 12-layer decoder-only transformer with masked self-attention heads that was pre-trained on the BooksCorpus dataset. This dataset contained over 7,000 unpublished books which is about 800M words. After pre-training, the model was fine-tuned on the Persona-Chat dataset using an augmented input representation to optimize a combination of two loss functions; multi-task learning. The Persona-Chat dataset consisted of 10k dialogues where each member in the conversation had a predefined persona. The augmented input representation combined 3 different token embeddings to create an input for the model. The first embedding was the word embedding which was the tokenized concatenation of the persona description sentences, dialogue history and any words of the output sequence that had already been generated. The second embedding was the positional embedding that contained the position information for each token in the word embedding. The third embedding was referred to as the Dialogue state embedding and indicated whether a word is part of persona sentence, a statement from person A in the conversation or a statement from person B in the conversation. The two loss functions being optimised were the next-utterance classification loss, which focused on classifying the correct next-utterance appended to the input sequence, and the language modelling loss, which focused on obtaining the next word in the sequence. To decode the outputs of the model a beam search with sampling and a small beam size of 4 was used [18].

This paper is relevant to this project as it provides a method on how to produce a state-of-the-art conversational chatbot which is a key component of this project. This method uses some of the prevalent trends in NLP currently with a pre-trained model and fine-tuning techniques that produce above average results. Some of the limitations of this paper is that it only considers using this method for one language, English. So to apply this technique further research would have to be completed to find the relevant datasets in Chinese, if there they are actually available. Another limitation of this paper is its use of beam search as the decoder. The more recent work of Holtzman et al. [19] shows that the distribution of words generated using beam search is very different from the distribution of words in texts created by humans proving that beam-search is unsuitable as the decoder in this instance. A more suitable option would be to use top-k sampling which was explored in the work of Fan et al. [20]

2.4.2 A Large-Scale Chinese Short-Text Conversation Dataset

The research paper "A Large-Scale Chinese Short-Text Conversation Dataset" won Wang et al. the International conference on Natural Language Processing and Chinese Computing (NLPCC) 2020 Best Student Paper Award [21]. Organised by the China Computer Federation(CCF) and the Technical Committee of Chinese Information (TCCI) ,the NLPCC provides a forum for researchers and practitioners from academia, industry, and government to share their ideas, research results and experiences, and to promote their research and technical innovations in their fields. Each year they call for original technical papers on new concepts, innovative research, systems, standards, testing & evaluation, applications, and industrial case studies related to NLP & CC [22]. This paper focuses on building a large-scale high-quality Chinese conversation dataset, using a custom data cleaning pipeline. This conversation dataset is then utilised in the experimentation of creating pre-trained models that can be fine-tuned for conversation[23].

In the paper, two Chinese conversation datasets were produced, LCCC-base and LCCC-large. To produce LCCC-base, 79M conversations were crawled from Weibo (a Chinese micro-blogging website) and were passed through the data cleaning pipeline. The pipeline consisted of 2 types of filtering: rule-based noise filtering and classifier-based filtering. The rule-based noise filtering defined a set of rules used to filter the conversations such as, remove the dialogues if 90% of trigrams in the response are high-frequency trigrams. The classifier-based filtering was used to filter the semantic, grammar and context-dependent noise that could not be captured with the rule-based noise filtering. To produce the LCCC-large, the79M Weibo conversations were mixed with multiple open-source corpora before passing them through the data cleaning pipeline, but with the classifier's threshold in the classifier-based filtering being relaxed.

These datasets were then used to create three pre-trained language models CDialGPT LCCC-base; a 12-layer GPT trained for 70 epochs on a Chinese novel dataset and post-trained for 30 epochs on LCCC-base, CDialGPT2 LCCC-base; a 12-layer GPT2 trained for 70 epochs on a Chinese novel dataset and post-trained for 30 epochs on LCCC-base and CDialGPT LCCC-large; a 12-layer GPT trained for 70 epochs on a Chinese novel dataset and post-trained for 30 epochs on LCCC-large. These pre-trained models were then fine-tuned for conversation using the STC dataset that contains 4.4M Chinese conversation pairs and had their performance evaluated against other generative chatbots.

In the paper, two forms of evaluation were performed: one using a set of language metrics and the other using 3 human judges. The results of the language metric evaluation are highlighted below.

Models	Size	PPL	BLEU-2	BLEU-4	Dist-1	Dist-2	Greedy Matching	Embedding Average
$\mathrm{Attn-Seq2Seq}$	$73M$	34.2	3.93	0.90	0.0085	0.1191	0.6584	0.8338
Transformer	$113M$	22.10	**6.72**	3.14	0.0088	0.1397	0.6606	0.8355
$\mathrm{GPT2-chitchat}$	$88M$	–	2.28	0.54	**0.0103**	**0.1625**	0.6154	0.7894
$\mathrm{GPT_{Novel}}$	$104M$	21.27	5.96	2.71	0.0080	0.1172	0.6612	0.8334
$\mathrm{CDialGPT_{LCCC-base}}$	$104M$	18.38	6.48	3.08	0.0083	0.1268	0.6621	0.8354
$\mathrm{CDialGPT2_{LCCC-base}}$	$104M$	22.76	5.69	2.50	0.0077	0.1087	0.6624	0.8346
$\mathrm{CDialGPT_{LCCC-large}}$	$104M$	**18.23**	6.63	**3.20**	0.0083	0.1271	**0.6625**	**0.8363**

Figure 4: Chatbot Automatic Metric Evaluation [23]

Referring to Figure 4 CDialGPT LCCC-large outperformed the other models in the 4 metrics of perplexity (PPL), BLEU-4, greedy matching and embedding average. Perplexity is a form of intrinsic evaluation meaning that it evaluates the actual language model itself rather than the language model's performance on a particular task. Perplexity is the inverse probability of a model's prediction on an unseen test set, normalized by the number of words. In layman terms it measures how surprised or perplexed the model is by sentences in the test set which conveys it level of understanding around how the language works. The CDialGPT LCCC-large model clearly outperformed the other models in this metric with it's closest competition being CDialGPT LCCC-base. BLEU stands for Bilingual Evaluation Understudy and provides a method of automatic machine translation evaluation based around n-gram matching between a model's translation and reference translation [24]. An n-gram is a sequence of n words and the higher number of matches between the model and reference translation the more accurate the model translation is. In this paper BLEU with 2-grams, BLEU-2, and BLEU with 4-grams, BLEU-4, were used during the evaluation. The Transformer model ranked first in the BLEU-2 metric with the CDialGPT LCCC-large model ranking a close second. However, the CDialGPT LCCC-large ranked first in BLEU-4. Greedy matching and embedding average are used to evaluate the relevance between the test data and the generated responses at the word and sentence level. Again, the CDialGPT LCCC-large model ranked first in both of these methods of evaluation. Next are the results of the human evaluation covered in Figure 5.

Models	**+2**	**+1**	**+0**	**Score**	**Kappa**
Seq2Seq	27.1%	21.4%	51.4%	0.756	0.4544
Transformer	42.4%	23.6%	34.0%	1.084	0.4429
GPT2 − chitchats	24.3%	19, 1%	56.6%	0.677	0.3941
$\text{CDialGPT}_{\text{LCCC-base}}$	46.7%	21.8%	31.5%	1.152	0.3954
$\text{CDialGPT}_{\text{LCCC-large}}$	**48.6%**	**24.5%**	**27.0%**	**1.217**	0.4203

Figure 5: Chatbot Human Evaluation[23]

200 samples from each model, 1000 in total, were presented at random to 3 judges who evaluated them using a 2/1/0 score schema. The score schema was based around fluency, relevance and informativeness with grammatically correct, logically consistent and relevant to the test data response getting a score of 1, fluent, relevant and additionally rich in content responses getting a score of 2 and anything else getting a score of 0. Again, the CDialGPT LCCC-large model outperformed all the other models receiving the highest percentage of 2 and 1 scores and the lowest percentage of 0 scores.

This paper is extremely relevant to the project as not only does it provide a method to create a generative chatbot in the desired target language, it also has made the pre-trained models open-source so they can be accessed by the NLP community. This provides a solid starting point to produce a state-of-the-art generative chatbot. This paper also incorporates the work of Wolf et al. [18] using a similar input representation combining 3 different token embeddings. However, it does not support personas which could be a possible limitation as the common issue with generative chatbots, a lack of a consistent personality, could be present.

2.4.3 Retrieve and Refine:Improved Sequence Generation Models For Dialogue

Another paper of interest was the work published by Weston et al. looking at improving generative chatbot models by incorporating retrieval chatbot models [25]. This paper aimed to address the weakness of generative models referred to as the “I don’t know” problem, also highlighted in the work of Wolf et al. [18], and the weakness of retrieval models, producing completely erroneous responses due to the message not existing in its retrieval set. The prototype model named “RetrieveNRefine” takes a standard generative model, in this case a 2-layer LTSM with attention, and concatenates the output of the retrieval model, in this case a Key-Value Memory Network, to its usual inputs. This model showed improved performance in producing more engaging response using more unique words while still maintaining consistent personas [25].

This paper is relevant for the project as it provides a good example of a hybrid chatbot model, an area that will be explored in this project depending on the time constraints as mentioned in the section 2.2 Chatbot Types. Like the “TransferTransfo” method [18] it isn’t directly applicable to the project’s problem area as the model produced isn’t for the desired target language. However, it does introduce an interesting model architecture that could be explored and developed further to improve the chatbot responses in the prototype system to better engage users.

2.4.4 Conclusion

Of the research papers investigated in this section, the paper "A Large-Scale Chinese Short-Text Conversation Dataset” provides an in-depth insight and also a readily available implementation of a Chinese generative Chatbot. This implementation fits the use case of the project so will be adopted and adapted for use in the prototype system

2.5 Summary

In this section two existing implementations of chatbots used in foreign language learning for Chinese were evaluated, highlighting their current strengths and weaknesses. The different chatbot type classifications were also covered prior to the optimal chatbot type for this project being defined. Moreover, the two most popular Chinese proficiency assessments were reviewed before a critical analysis of research paper relating to the project was completed. The research performed in this section will be instrumental in determining the requirements of the proposed system, detailed in the next section.

3 Systems Requirements and Specification

This following section will outline the expected functionality of the prototype system in the form of functional and non-functional requirements. The functional requirements will specify the behaviour and functions of the system with their priorities being defined using the MoSCoW prioritisation technique. The non-functional requirements will specify how the system will perform certain functions.

3.1 Project Scope

After conducting the literature review, it is apparent that the scope of this project is vast. Each aspect of the prototype system: a chatbot system, a classification system, a NLP system and a improvement recommendation system when applied to the Chinese language domain could be individual projects in their own right. Due to the time, personnel and resources constraints, it is unlikely that the full system can be implemented. So, to ensure that there is some form of implementation that can be evaluated at the end of project, the scope will be managed using a minimum viable product (MVP). For this system the MVP would be a chatbot system that, through conversation in the form of messages, determines a user's proficiency in Mandarin Chinese. This system will still address the problem area of the project as it will provide users with a proficiency level for their current active vocabulary. In most cases, especially when learning a second language, there will be a large gap between a person's passive vocabulary, the words a person knows but does not know how to use, and their active vocabulary, the words a person knows and uses [26]. Through providing a proficiency assessment of a user's active vocabulary the user will be able to assess the size of the gap between their active and passive vocabulary. They will then be able to practise lessening that gap through conversations with the chatbot. The same also applies for the grammar patterns. If there is still time available after the implementation of the MVP, adding features from the original project scope will be explored. This is also covered in the section 3.2 Functional Requirements .

3.2 Functional Requirements

All the functional requirements listed will contain the following information:

- An ID, which will act as a reference for the requirement in future sections
- A name
- A description of the requirement
- A description of its inputs
- A description of its outputs

3.2.1 Must Requirements

In the MoSCoW prioritization categories must-have initiatives are the non-negotiable, high-priority requirements that must be implemented, otherwise the system would be seen as inoperable [27]. The "Must" requirements will represent the requirements needed to produce the minimum viable product mentioned in section 3.1 Project Scope.

ID	Name	Description	Input	Output
FM1	A user must be able to register an account	A user must be able to register an account on the system so they can	The user inputs of: their full-name, a unique username,	A user account will be created and stored on in the application

		login	email and password	database
FM2	A user must be able to log into the system	A user must be able to user the account information they used to register an account to log into the system	The account username and password	The user will be logged into the system and gain access to the full system functionality
FM3	A user must be able to log out of the system	Once a user is down using the system they must be able to logout of their account	Pressing a logout button	User is logged out of system and redirected to the login page
FM4	A user must be able to converse with the chatbot	A user must be able to send messages to the chatbot in Chinese and receive replies in Chinese	A message in Chinese entered by the user	A message in Chinese generated by the chatbot based on the user message received
FM5	A user must be able to see information on the words used in their messages	A user must see a real time list of all the words they have used in their messages to the chatbot and their corresponding proficiency level	User messages	A list of unique words used in the user messages with their corresponding proficiency level
FM6	A user must be able to see information on the grammar patterns used in their messages	A user must see a real-time list of all the grammar patterns they have used in their messages to the chatbot and their corresponding proficiency level	User messages	A list of unique grammar patterns used in the user messages with their corresponding proficiency level
FM7	A user must be able to see the average proficiency level of their vocabulary	A user must see the average proficiency level of the words they have used in their messages to the chatbot	The list of unique words used in the user messages with their corresponding proficiency level	The average proficiency level of the words

FM8	A user must be able to see the average proficiency level of their grammar	A user must see the average proficiency level of the grammar patterns they have used in their messages to the chatbot	The list of unique grammar patterns used in the user messages with their corresponding proficiency level	The average proficiency level of the grammar patterns

Table 2: Must Requirements

3.2.2 Should Requirements

In the MoSCoW prioritization categories should-have initiatives are the requirements that are not vital to the system, but whose presence add significant value [27]. For this project, the “Should” requirements encompasses the the proficiency improvement suggestions functionality from the original project scope and additional user account functionality.

ID	Name	Description	Input	Output
FS1	A user should be able to see word and grammar information from previous conversations	A user should be able to see a list of words and grammar patterns with their corresponding proficiency level from their previous conversations with the chatbot	The account information of the user requesting their conversation history	A list of a conversation
FS2	A user should be able to change their account information	A user should be able to change their full-name, username, password and email	The new account information for a user	The updated user account
FS3	User accounts should be verified by email	Once a user registers they should receive an email asking them to verify their account	Email for unverified user account	Verified user account
FS4	A user should be able to delete their account	A user should be able to delete their account and all it’s associated information from the system	The account’s password to verify the delete operation	The user account and it’s associated word and grammar information is removed from the system
FS5	A user should receive suggestions on how to improve	A user should receive suggestions on how to improve their Chinese	The word and grammar information from	A list of suggestions on how to improve their Chinese

	their Chinese proficiency	proficiency after an analysis of their conversation with the chatbot	the user's conversation with the chatbot	proficiency

Table 3: Should Requirements

3.3 Non-functional Requirements

All the non-functional requirements listed will contain the following information:

- An ID, which will act as a reference for the requirement in future sections
- A description of the requirement

ID	Description
N1	The user will access the system through a web client
N2	Response from chatbot within 5 seconds
N3	User input analysis within 25 seconds
N4	All user requests will be authenticated

Table 4: Non-functional Requirements

The non-functional requirement N2 is listed to ensure that the user experience is not hinder by slow responses from the chatbot, which will allow the conversation to feel more natural and flow. The requirement N3 has the time limit of 25 seconds as any user input analysis must be performed before a HTTP timeout. N4 will be implemented using tokenization which will be discussed later section 4.5 Security Design.

3.4 Software Development Lifecycle Methodologies

The Software Development Lifecycle (SDLC) defines the processes used during the development of software or systems. The goal of the SDLC is to produce the highest quality and lowest cost software or system in the shortest amount of time. The areas it focuses on are: requirement analysis, planning, software/architectural design, software development, testing, and deployment. There are multiple SDLC models that apply these phases in different ways, with the some popular models being the Waterfall, Iterative and Agile models [28].

The Waterfall model is one of the oldest SDLC models which strictly follows the above phases in order. Each phase must be completed before the next one can be started ensuring that none of the phases overlap. An advantage of this model is its simplicity, making it clear and easy to understand. However, a major disadvantages is its lack of flexibility, for example no changes can be made to the project requirements until the end of the entire development cycle [28].

The Iterative model is based on repetition. An initial simple implementation of the software requirements is created before iteratively enhancing the features and introducing new ones until the full system is implemented. An advantage of this model is that working functionality can be developed early in the lifecycle, but a major disadvantage is that resources can become depleted very quickly [28].

The Agile model splits the system into cycles and delivers new functionality each cycle. This creates a succession of software releases with each release being tested. This model enables the development team to produce a working system relatively fast and the feedback after each release is used to improve

the next release. Some advantages of this model include frequent releases resulting in faster error diagnosis and correction and a quick turnaround for feedback from end users. A disadvantage of this model is that due to the constant requirement changes it can be difficult to determine the resources needed for the project.

After considering the different SDLC models, the model that best fits this project is the Agile development model. It supports the previously defined work plan as each aspect of the system can be developed each cycle and it provides flexibility around some the development uncertainty associated with the project. It is also the model that fits best with the time constraints of the project.

3.5 Summary

In this section the functional and non-functional requirements of the prototype system were laid out and an appropriate SDLC model was chosen. These requirements were used to create designs for the prototype system, documented in section 4 System Design.

4 System Design

In this section the design of the prototype system will be presented. First, the architecture of the different machine learning models that will be used in the system will be reviewed. Next, the high-level behaviour of the system will be displayed in the form of use case and activity diagrams. Following this, the database designs will illustrate how user information will be stored with user-interface designs providing a reference for how the prototype system will look. Decisions such as system architecture will also be presented and discussed.

4.1 Machine Learning Model Architectures

In the prototype system two Machine learning models will be utilised, the first being the chatbot model and the second being the classification model. In this section we will review the architecture of these models

4.1.1 Chatbot Model

The CDialGPT LCCC-large model will be adopted from the work of Wang et al [23], previously discussed in section 2.4.2, and used as the chatbot model in the system. The architecture of this model is based on the Generative Pre-trained Transformer (GPT) developed by Radford et al. [29]. Figure 6 below displays an illustration of the GPT model architecture taken from their paper.

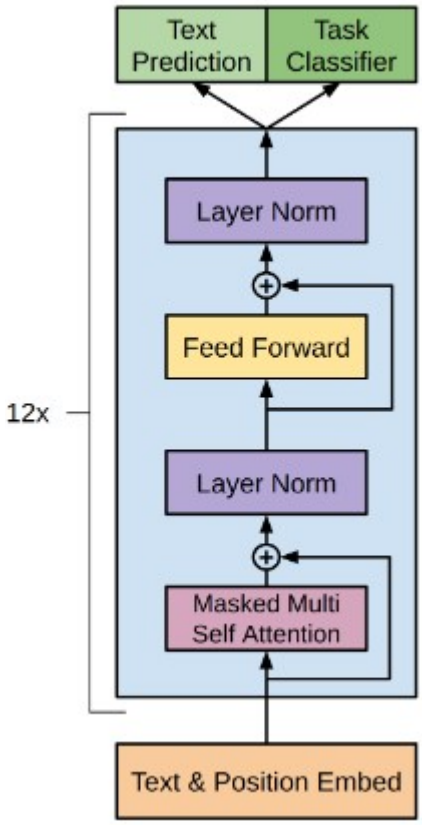


Figure 6: GPT Model Architecture [29]

The GPT model is a 12-layer decoder-only transformer with masked self-attention heads and Figure 6 highlights the contents of one of those transformer decoder layers. This model is derived from the

decoder part of the original encoder-decoder Transformer of Vaswani et al. [30]. Each decoder layer takes tokenized text concatenated with the position embeddings as the input and passes them first through the masked multi self-attention layer. Tokenized means that the words in the text were converted into a number representation based on the model's vocabulary; the list of unique words in the text that the model was pre-trained on. The masked multi-self attention layer is made up of multiple scaled dot-product attention blocks, in the GPT model's case 12, stacked in parallel as seen in Figure 7 below. Self-attention is the process of the model performing a set of matrix multiplication calculations prior to applying soft-max and scaling functions. This determines which tokens (words) in a phrase are relevant to other tokens in the phrase, and thus which tokens in a phrase are important. Masked attention means that the Transformer only has access to the subsequent tokens to the left of current token when performing this process. The output of this layer is then added to the residual connection, represented by the ⊕ symbol in Figure 6, which is the tokenized text and position embeddings before being passed through the masked multi self attention layer. After this addition, layer normalization is performed, with both of these measures stabilizing the training process. The result of the residual addition and layer normalization is then passed the feed forward layer which is applied position-wise to each token passed along. The feed forward layer consists of two linear layers and one ReLU activation layer. Finally the residual connection, in this case the output of the previous layer normalization, is added to the output of the feed forward layer before again being normalized producing the final output [30].

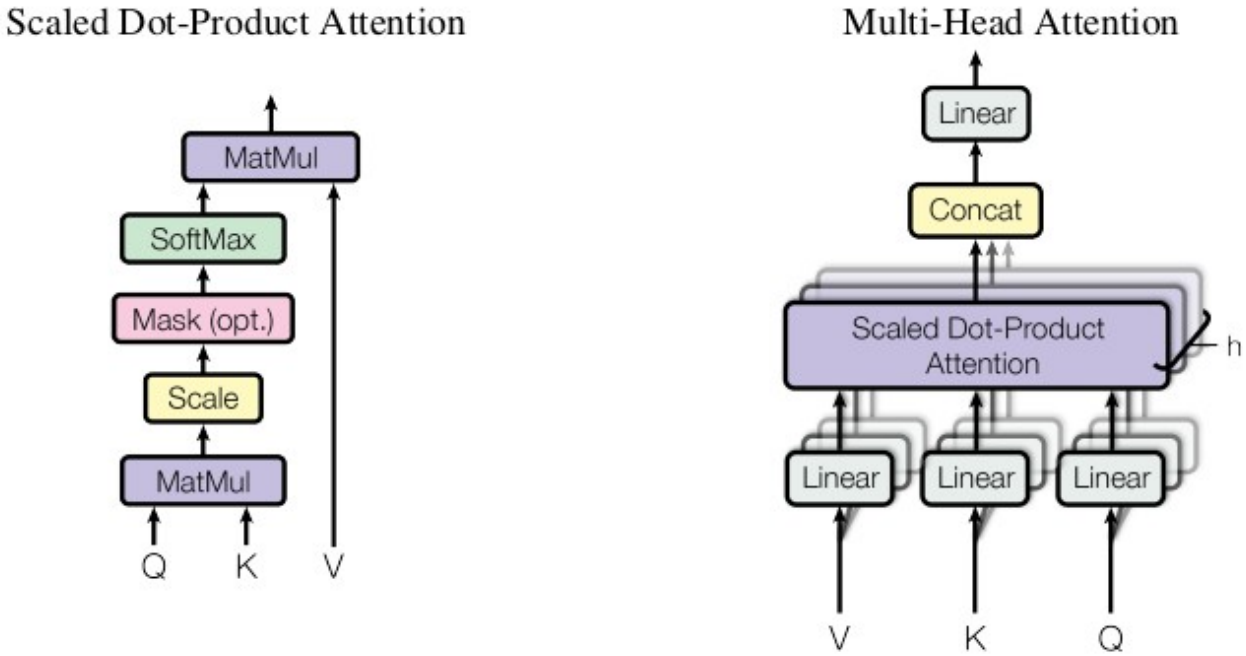


Figure 7: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention[30]

4.1.2 Classification Model

Like the chatbot, a pre-trained model will be fine-tuned on a dataset, in this case one of the vocabulary lists for the proficiency exams discussed in section 2.4, to produce a classifier for the task of assigning a Chinese word a level of proficiency. Fine-tuning is an example of transfer learning; a machine learning approach where a model developed for one task is reused and applied to another task within a similar domain [31]. To decide on which pre-trained model to use for this task, a set of models will be

selected for the Chinese Language Understanding Evaluation (CLUE) Benchmark leader board and tested. The top performing model will then be chosen and incorporated into the prototype system. CLUE is the Chinese equivalent of GLUE [32], providing a benchmark for evaluating the performance of Chinese natural language understanding (NLU) models in the form of 9 different natural language understanding tasks [33]. A popular pre-trained model that appears on the CLUE leader board is Bert including some of it's variations like Roberta and Albert.

Like the GPT model, Bert is also derived from the encoder-decoder Transformer of Vaswani et al. [30], except it is made up of several layers of encoders rather than decoders. The Bert base model has 12 encoder layers which receive tokenized text as their inputs. The encoder layer architecture is almost identical to the GPT decoder architecture displayed in Figure 6, but the masked multi self attention layer is replaced with a multi-head attention layer. This means that the masked optional layer in the scaled dot-product attention blocks (refer to the left part of Figure 7) is not included. For the classification task an extra linear layer will be added to the model architecture to map the model outputs to the set of classification classes.

4.2 Use Case Diagram

Use case diagrams portray the interactions between a user and a system. The use cases capture the functionality of the system, actors represent users of the system, and the associations between a use case and actor represents the actors ability to utilise that certain system functionality.

Figure 8 displays how users will be able to interact with the MVP system with all users being able to: register an account, login, converse with the chatbot in Chinese and view the proficiency level of the words and grammar patterns they are using in that conversation.

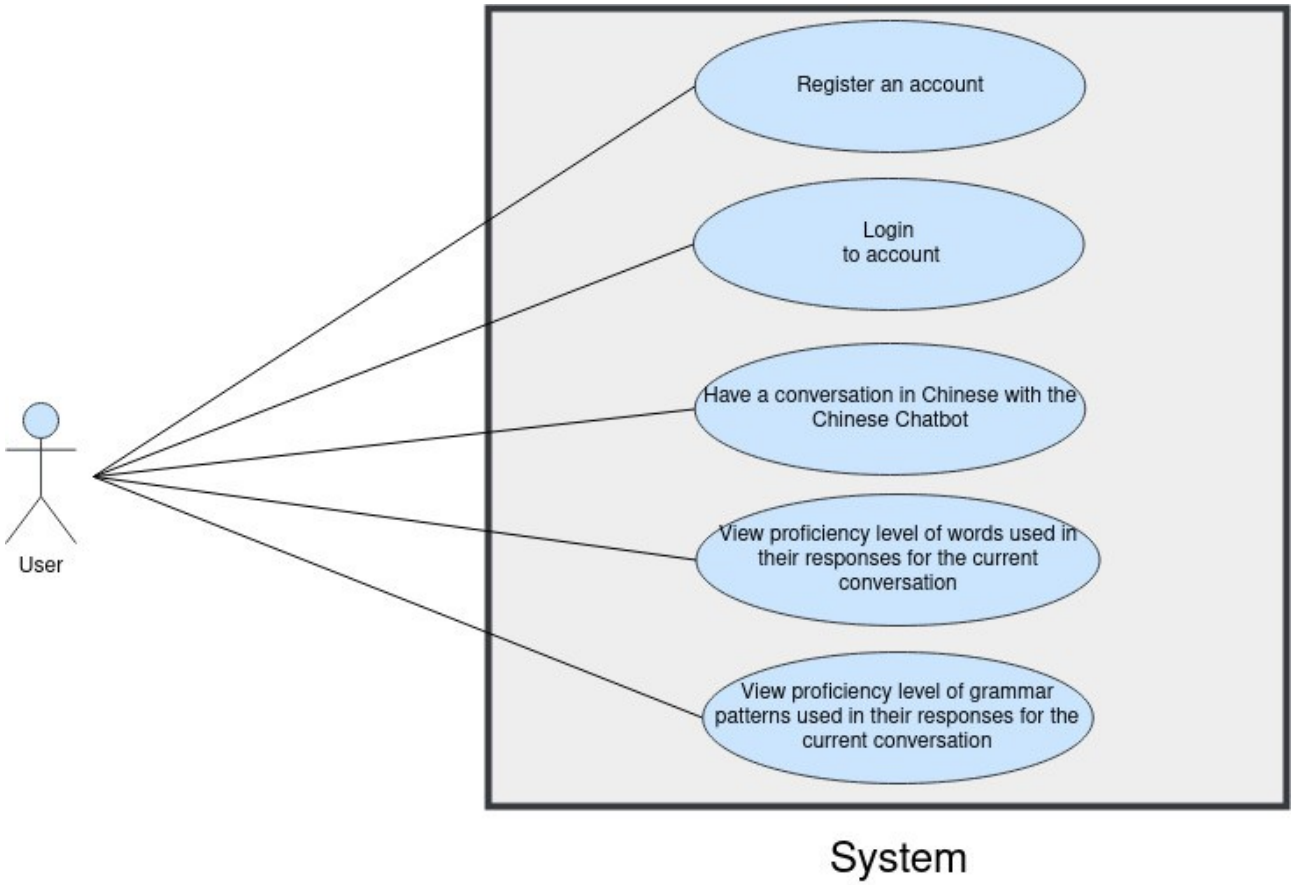


Figure 8: MVP Use Case Diagram

Figure 9 exhibits how users will be able to interact with the system including the additional functionality of FS1, the ability to view the grammar and word information from previous conversations they have had with the chatbot

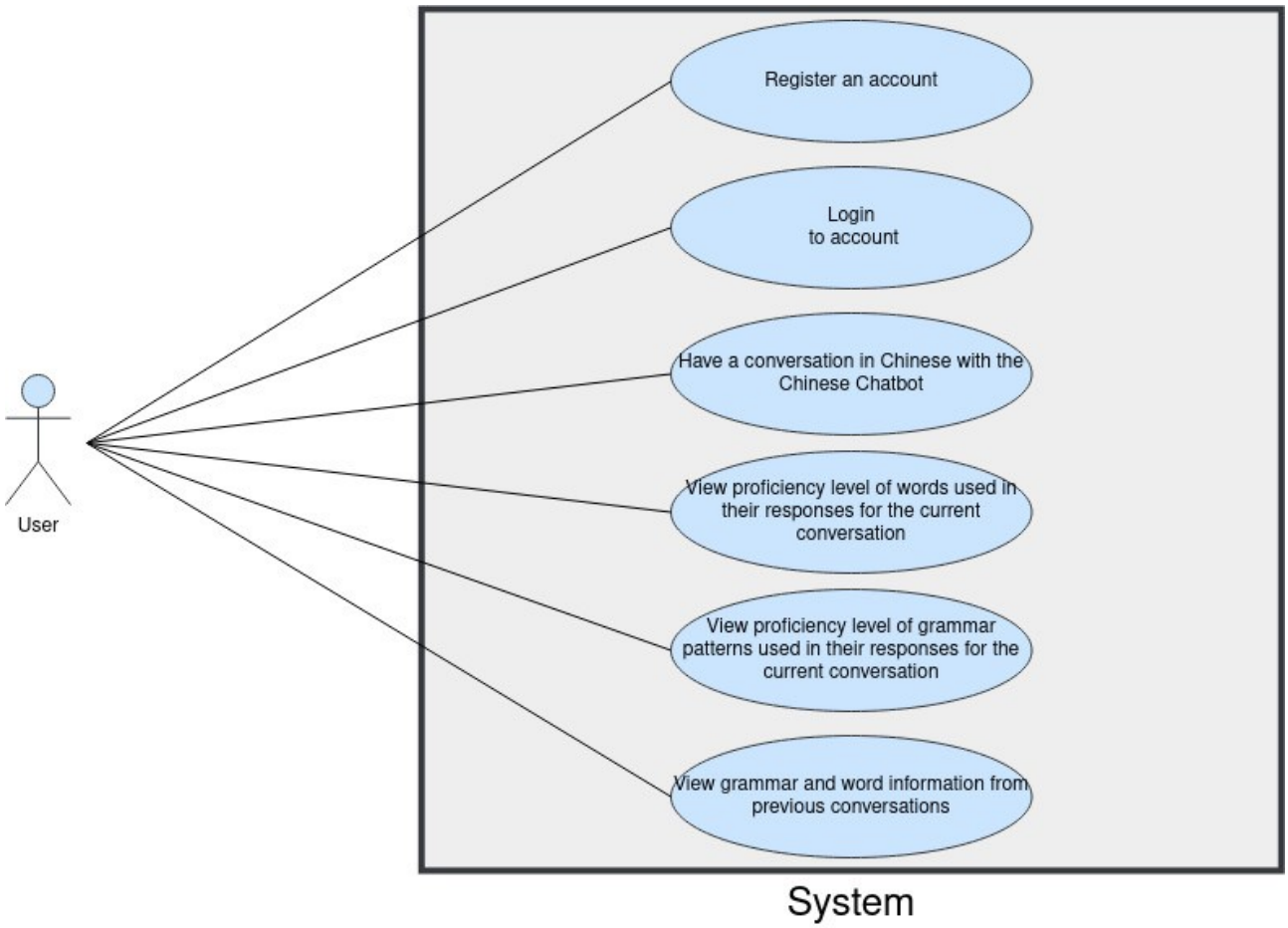


Figure 9: Use Case Diagram with FS1 Requirement

4.3 Activity Diagrams

Activity diagrams are another type of UML diagram. The use case diagram provides a high-level view of the system functionality, with the use cases only supplying a brief description. Activity diagrams facilitate the detailed description of the process taken during an individual use case with ability to also display the interactions between different use cases.

Figure 10 and 11 are examples of how use cases can be broken down using activity diagrams. The registering an account use case from Figures 8 and 9 is expanded upon in Figure 10 and the login to account use case is covered by Figure 11. All the other use cases were also broken down in a similar manner .

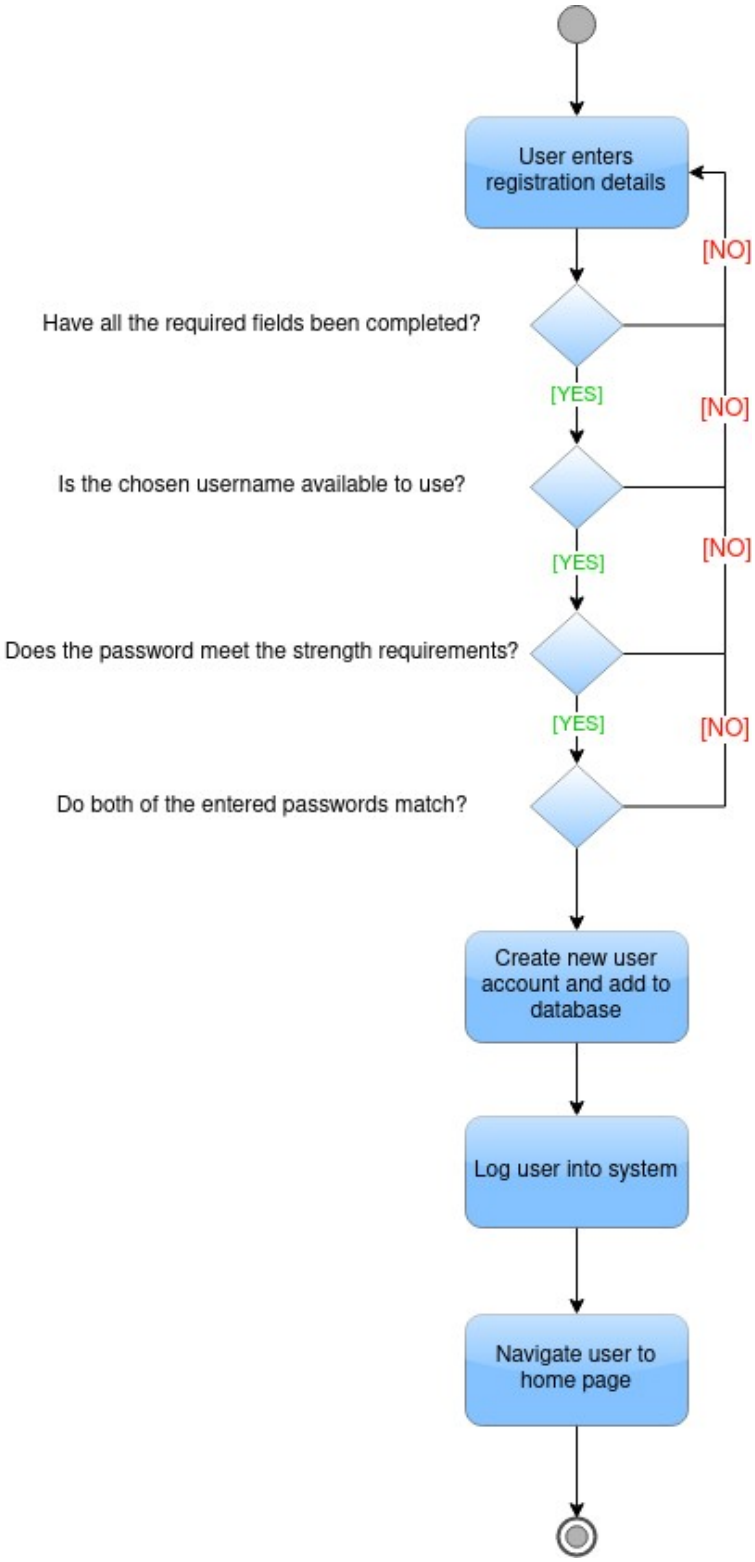


Figure 10: Activity Digram showing a user registering a new account

In Figure 10 the multiple stages of validation that is carried out by the system when a user is registering a new account becomes evident. First, the system checks all the necessary information to create a new user was entered. Next, the system checks the entered username of the new user is not already in use, all usernames must be unique. After that, the system checks that the password entered meets the system's password strength requirements. Finally, the system checks that the password and confirm

password entered match; used for password confirmation. If all the stages of validation are passed a new user is added to the system, the user is logged in and navigated to the home screen

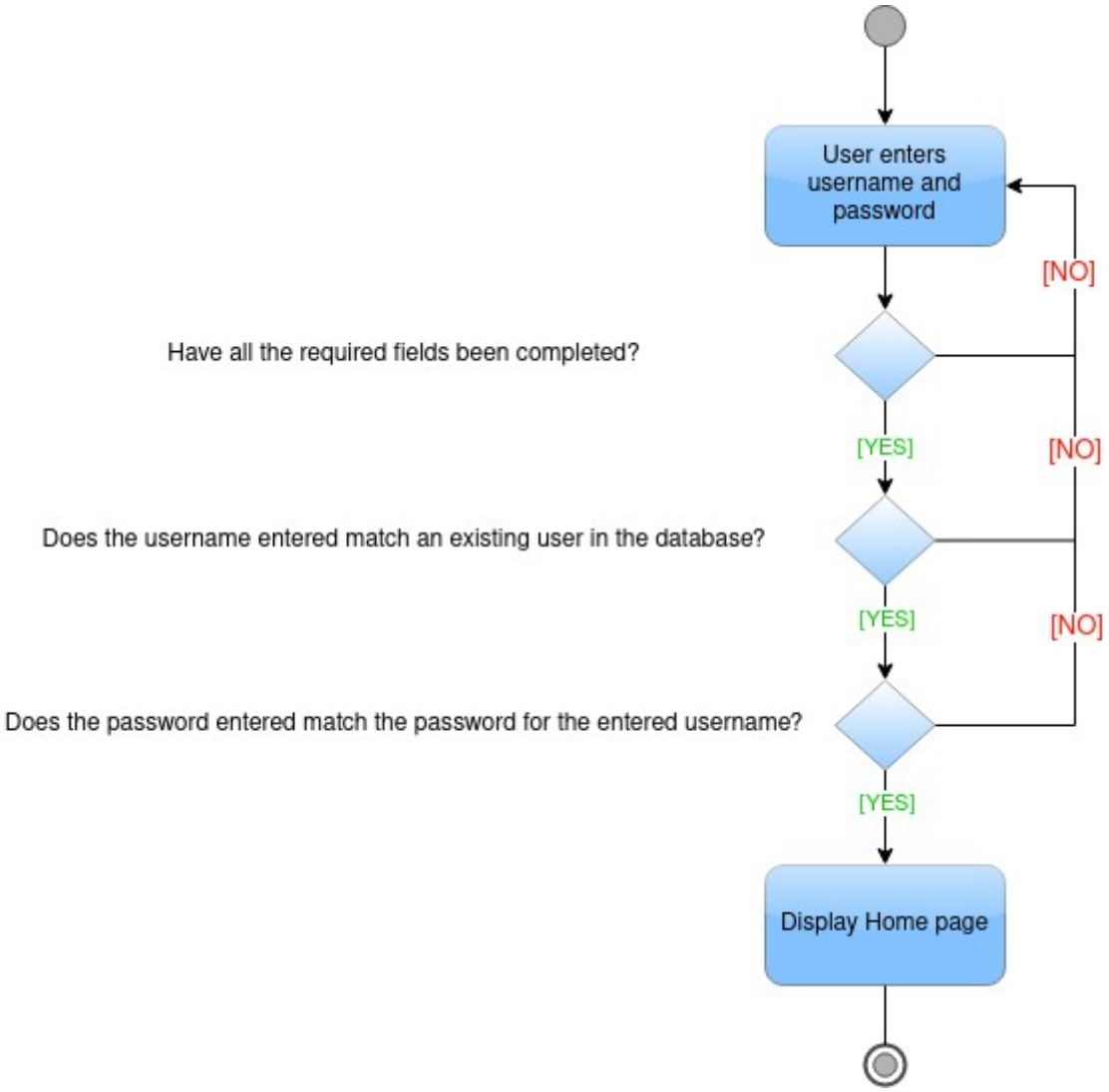


Figure 11: Activity Diagram showing user logging into the system

Figure 11 also displays the multiple stages of validation performed by the system during the login process. The system checks that all the necessary login information is entered, the entered username exits in the system and the password entered is correct. After satisfying all the stages of validation the user is navigated to the homepage

4.4 System Architecture Design

The prototype system will be developed with a micro-service architecture making the application a set of loosely coupled, collaborating services. According to O'Reilly's Micro-services Adoption in 2020 Report [34], 77% of businesses have now adopted micro-services clearly showing the architecture's growing popularity. This is further highlighted by the findings of The Software House's State of Micro-services 2020 report, with 85.5% of developers surveyed stating that the micro-services architecture will become the industry standard. This is a sentiment also personally shared after completing an industrial placement and observing this architecture's capabilities first hand. The micro-service architecture is the most suitable for this project as the system is a combination of different parts each with their own software requirements. This architecture ensure there are no dependency conflicts and

will allow all the different pieces to slot together and collaborate to produce the final system. This also compliments the agile software approach that will be taken, as during each sprint a different micro-service can be developed.

The technology that often accompanies the micro-service architecture is Docker and that technology will also be used in the development of the prototype system. Docker is software that builds and delivers applications based on containers [35]. Containers are units of code bundled with their specific dependencies that run in isolation to each other, but all make shared use of the operating system kernel they are running on. This approach has several benefits, the first main benefit being the flexibility to run the code on any machine, regardless of its individual configuration or infrastructure, and have it always run the same. The second main benefit of containers is due to the shared use of a single operating system kernel; they use fewer resources than virtual machines [36]. For each micro-service in the prototype system a custom docker file will be written, which is a definition of the software and files needed for that micro-service. Each docker file will then be built producing a docker container image, which is all the necessary code and dependencies bundled together that creates a docker container at runtime. Docker-compose, a command line interface for defining and running multi-container Docker applications, will then be used to define the network of docker containers, therefore allowing the different micro-service to communicate with each other during runtime.

Another advantage of Docker and the micro-service architecture is its compatibility with cloud technology, with the major cloud platforms like AWS, Google Cloud and Microsoft Azuer offering Docker support for application deployment. Combining this with orchestration tools like Kubernetes and Docker Swarm; software for automating deployment, scaling and management of containerized applications, and developing the prototype system in this manner has the added benefits of easy future deployment of the system that can be managed at scale. Figure 12 illustrates the architecture of prototype MVP system

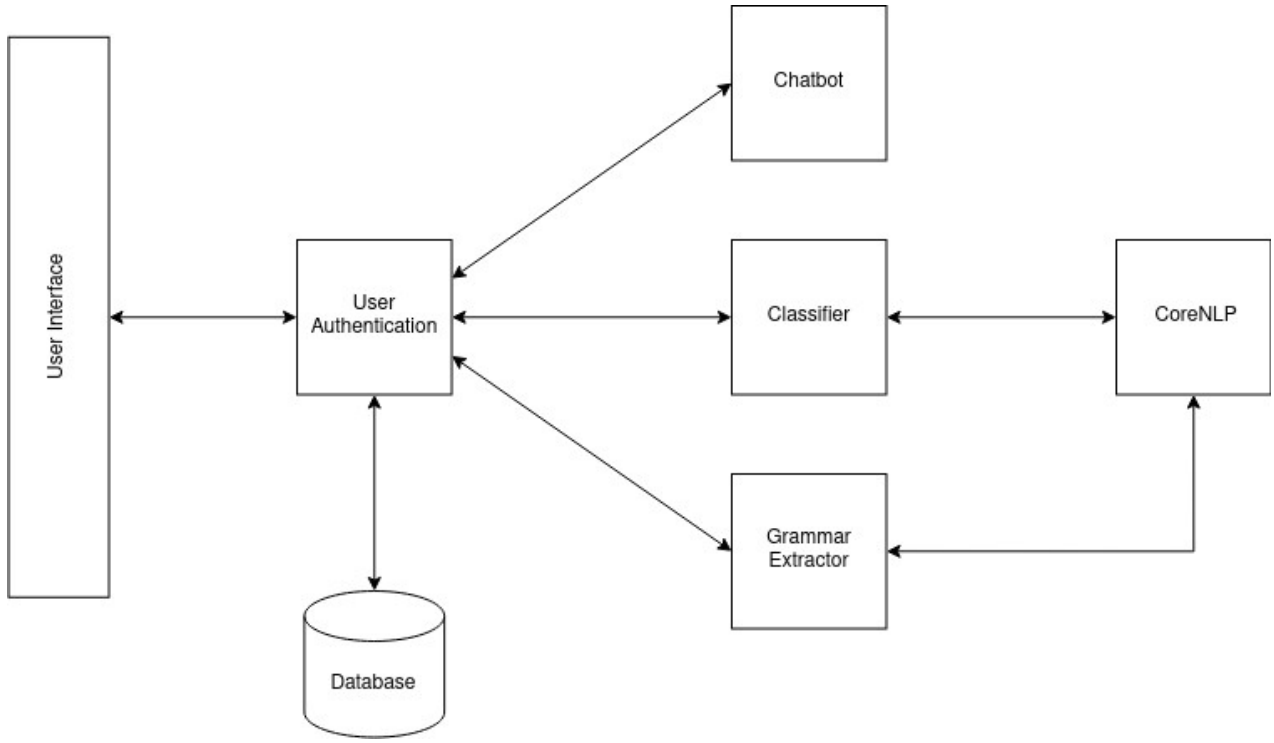


Figure 12: MVP Prototype Micro-service System Architecture

The prototype MVP system will be made up of 7 different micro-services: the User interface (UI) service will be responsible for displaying the user interface, the User Authentication service will handle authenticating user requests before forwarding them to the other services, the Chatbot service will hold the chatbot functionality of the system, the Classifier service will contain the classification model used to classify the proficiency level of words in user responses, the Grammar Extractor service will consist of the functionality to extract grammar patterns from text , the coreNLP service will provide services such as tokenization and POS tagging to the Grammar Extractor and Classifier services and finally the database will store the necessary application information. The User Authentication service will read from and write to the database during user registration and login. Figure 13 introduces the architecture changes that will be made if the additional FS1 requirement is implemented

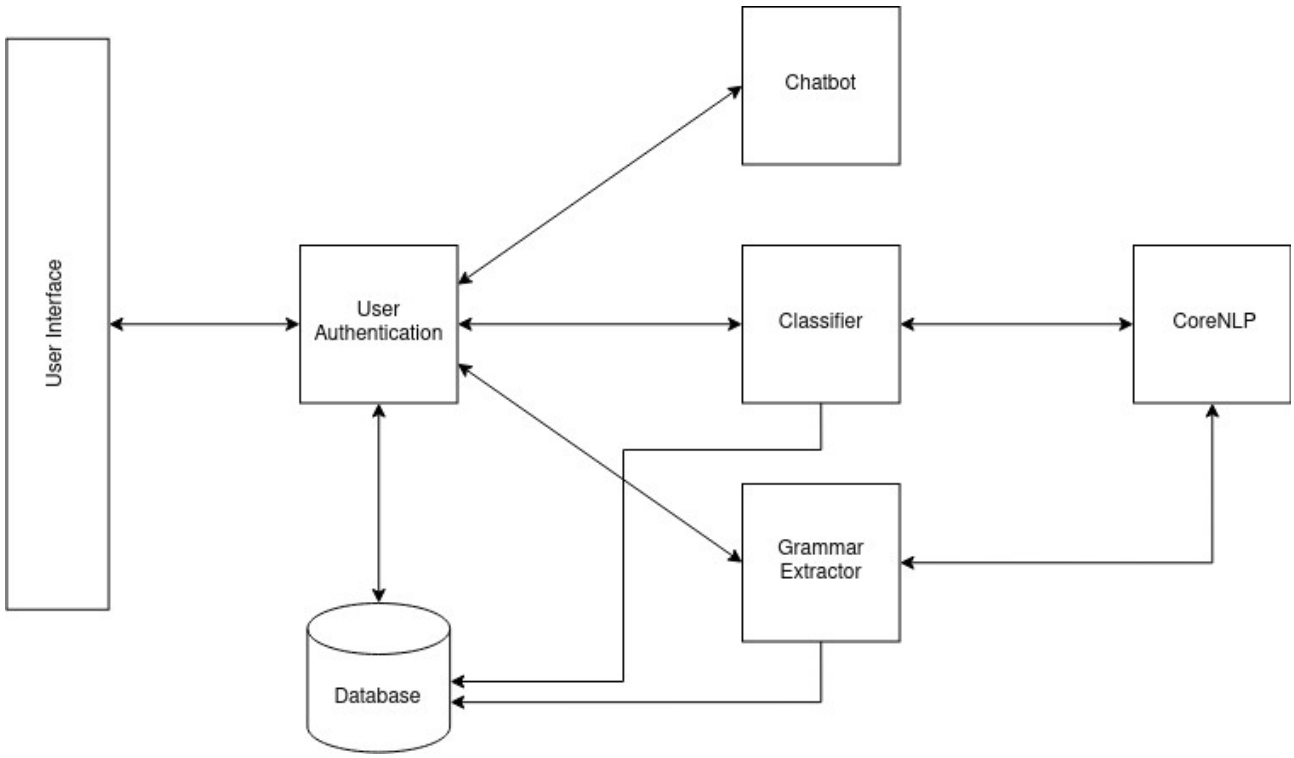


Figure 13: Prototype Micro-service System Architecture with FS1 Requirement

The number of micro-services has not changed but now the Classifier and Grammar Extractor services will also write to the database. This is to store word and grammar pattern information extracted from user responses during a conversation with the chatbot. The User Authentication service will then be tasked with retrieving those conversation with their associated words and grammar patterns.

4.5 Security Design

This section explains the security functionality of the system that will be implemented through the User Authentication Service (UAS), fulfilling the non-functional requirement N4.

As Figures 12 and 13 shows all user requests from the React UI service will be sent to the UAS for authentication. This authentication process will be carried out using tokenization, which is the process of transforming meaningful data into random strings of characters protecting the information against data breaches. For the prototype system JSON Web Tokens (JWTs) will be leveraged using the methods outlined by Gopal and Novick in their web article [37]. A JWT is JSON object which is used to securely transfer data between parties over the internet. It is made up of three parts: the JSON Object Signing and Encryption (JOSE) header; contains the metadata about the token type and cryptographic

algorithm used to protect the JWT contents, the JSON Web Signature (JWS) payload; a set of claims which is the content that will be protected and the JWS signature; used to check that the JWT is trustworthy and has not been tampered with during transmission [38]. The UAS will generate a JWT access-token with an expiration as a response to a user registering an account or logging in, with the JWT needing to be persisted on the client-side. Any following user requests will then include the stored JWT in the header which will allow the UAS to check that the request came from a genuine user. The UAS verifies the JWT by: checking it is well formed, checking the signature and checking the standard claims. A valid JWT will result in the user request being accepted and forwarded onto the relevant service. An invalid or expired JWT will result in the request being rejected and the user having to log back into their account to generate a new token.

Persisting the JWT in the local-storage of the browser, or in a cookie created on the client-side presents a security risk. Both of these can be accessed by any JavaScript code, making them prone to Cross-Site Scripting (XSS) attacks. XSS attacks are carried out by malicious users through injecting JavaScript code into an authorised user's browser. This code enables the malicious user to compromise the connection between the authorised user and system, allowing them masquerade as the authorised user and modify or access data which they are unauthorised to see. Storing the JWT in a HttpOnly cookie created by the UAS would defend against XSS attacks but, would still be vulnerable to Cross Site Request Forgery (CSRF) attacks. CSRF attacks allow a malicious user to send malicious requests using the credentials of an authorised user; usually a state changing request like changing an account password. To avoid all these attacks the JWT will simply be stored in memory.

Generally speaking, the access-token stored in the browser should expire after a short period of time, in the prototype system 15 minutes, resulting in a user being forced to log in quite frequently to re-authenticate. Storing the access-token in memory also introduces a different problem of the token being deleted if the user refreshes the page or switches tabs in the browser. Again, resulting in the need for re-authentication. These issue would greatly affect the overall user experience of the prototype system so to resolve them and also facilitate the persistence of a user session a JWT refresh token will be used.

A refresh token is a JWT with a longer expiration than the access-token that will also be sent in the response to a user logging in or registering an account. However, it will be stored in the browser as a Http only cookie instead of in memory. This cookie will then be use to query a specific "refresh" endpoint that will return a new access-token. Although the refresh cookie is technically safe from CSRF attacks, even though an attacker can make a request to the refresh endpoint there is noway for the them to acquire the fresh access-token that is returned, another layer of protection has been added in the form of the double submit cookie pattern. The double submit cookie pattern works by sending a CSRF token cookie containing a random string along with the refresh token cookie, except that this CSRF token cookie is not Http only. On the client side when making a request to the "refresh" endpoint, as well as sending the refresh token cookie, the contents of the CSRF token cookie has to be extracted and included in the header of the request otherwise the request will be rejected.

This will resolve the issues mentioned through silent refreshes. This is a request that is automatically sent to the refresh endpoint when either the stored access-token is about to expire or when the user first access a web page of the prototype system. The first silent refresh removes the need of the user having to re-authenticate on access-token expiration. The second silent refresh handles the issue of storing the access-token in memory, while also persisting a user session for as long as the refresh token remains valid. This will allow the system to provide a good user experience and still remain secure.

When creating an account on the system the user will have to create a password that adheres to the password strength standards set by NIST (National Institute of Standards and Technology) [39]; be at least 8 characters in length. The passwords will also be hashed before being stored in the database, another NIST standard.

4.6 Database Design

To implement the MPV use cases of registering user accounts and users logging into the system a database will be needed to persist the relevant data. Following the Microsoft database design basics , the key information that needed to be stored was found and organised. In the case of the MVP system this key information was user information. This information was then used to create the major entity of Account with this entity being used to create the database Tables. Account was chosen over User as the word user is reserved in SQL. For this entity, the information associated was stored in a Table, with each item becoming a field and being displayed as column in the Table. Taking the Account Table, an example of a field would be username or password. Next a primary key for the Table was then chosen which acts a unique identifier for each row. Finally, normalization rules were applied to ensure that the Tables were structured property and there was no redundant data.

When introducing the FS1 requirement of users viewing word and grammar information from previous conversations to the system, the same Microsoft database design basics process was followed. The additional key information that needed to be stored was information about the grammar patterns and words used by users during conversations. This information produces the new major entities of Grammar and Word with these entities again being used to create the database Tables. Again, for each of these entities the information associated with them was stored in the Table, with each item becoming a field and being displayed as column in the Table. Next a primary key for each Table was chosen which acts as a unique identifier for each row. Following this the relationships between the different Tables were specified through either adding new fields to the existing Tables or creating new Tables. In this case a new conversation Table was added which can be seen in Figure 15. Finally, normalization rules were applied to ensure that the Tables were structured properly and there was no redundant data.

4.6.1 Entity Relationship Diagrams (ERDs)

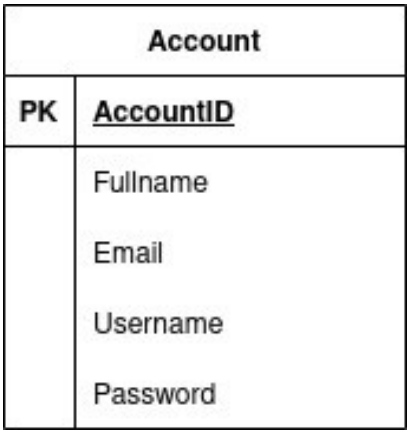


Figure 14: Entity Relationship Diagram for MVP system

Figure 14 displays the extremely simple ERD needed for the MVP system database. For the use cases of the MVP a single Table is needed to store user account information which contains the fields Full name, Email, Username and Password. The Email field is present to accommodate the “Should” requirement FS3.

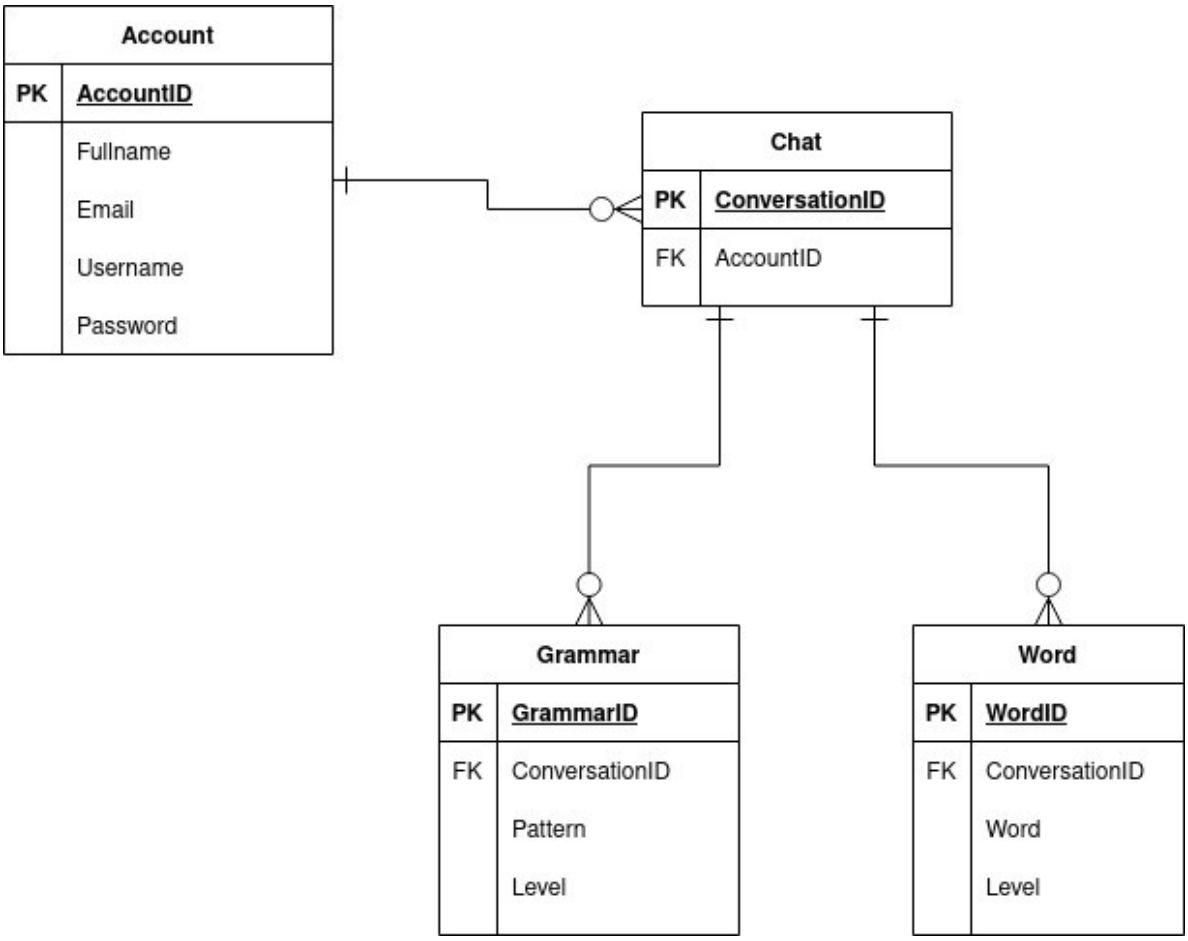


Figure 15: Entity Relationship Diagram for prototype System with FS1 Requirement

Figure 15 denotes the relationships and cardinality of the relationships between the different Tables in the database of the prototype system with the FS1 requirement. Primary keys are denoted as PK and foreign keys are denoted as FK. Each user account can have many conversations, signified by the one-to-many relationship between the Account and Conversation Tables. This relationship is based around the foreign key Account ID in the Conversation Table, which is also the primary key of the Account Table. A user account can use many different grammar patterns in a conversation. This is shown through the one-to-many relationship between the Conversation and Grammar Tables, based around the Conversation Table's primary key, ConversationID, appearing as a foreign key in the Grammar Table. The same one-to-many relationship also exists between the different words used by a user account in a conversation, so is implemented in the same way between the Word and Conversation Tables.

4.6.2 Database Management Systems

Database Management Systems (DBMS) are the pieces of software responsible for creating and managing databases. There are multiple DBMS available and a subsection of those will be evaluated for use in the prototype system, with the most suitable option being chosen. The DBMS considered can be found in Table 6 below, with the choices being the top 4 ranked DBMS according to a list of the most popular DBMS. The list's rankings are based not only on the usage among businesses, but also the volume of mentions and 'general interest in the system' on search engines, popular social networking sites such as twitter and well-known professional networking sites such as LinkedIn.

	Oracle Database	**MySQL**	**Microsoft SQL Server**	**PostgreSQL**
Provider	Oracle	Open-source	Microsoft	Open-source
Advantages	Oracle created a database that runs on any operating system Oracle incorporates Flashback technology. In the event of an application outage this technology will recover any incorrectly deleted or lost data Oracle also supports grouping transactions which gives it the ability to group multiple transactions into a batch for processing giving it greater scalability. Because Oracle Database is an IBM offering, users have access to the IBM support team	MySQL is easy to install, and has many third-party tools that make setting up simple MySQL has a large community of developers meaning there is a lot of support available MySQL is platform-independent MySQL is considered one of the fastest databases available. Moreover, it also provides multi-threading to achieve more optimized performance.	Microsoft SQL Server offers the flexibility to choose which components, such as cluster support, or business analysis are installed. Supports a range of data types, including JSON and XML. Microsoft SQL Server has a number of features that promote data restoration and recovery. Microsoft SQL Server ‘Always Encrypted’ feature ensures security of data	The first DBMS to implement multi-version concurrency control (MVCC). Highly customizable with the ability to define your own data types. PostgreSQL is fully SQL-compliant. Having a large open-source community improves the time to identify and resolve code errors. This also means there are many third-party applications, tools and integrated libraries to work with PostgreSQL.

Disadvantages	The cost of operating Oracle Database is high as it comes with a compulsory licencing fee . Oracle SQL is seen as being more difficult to learn and operate compared to other SQL competitors .	MySQL can be a little less reliable than other relational database management systems and tends to corrupt data under certain use cases. MySQL fails to implement the full SQL standard. This means it is not fully SQL-compliant MySQL does not provide adequate support for reading/write operations so has poor performance under high loads	Although use for development or educational purposes is free, the Standard version licencing fee is extremely expensive. Microsoft SQL Server licensing can be quite difficult to understand and is always changing.	Is not as popular as other database management systems such as MySQL. For simple, read-heavy workflows, PostgreSQL does perform as well as other open-source databases.

Table 5: Evaluation of DBMS

Table 5 shows that all the DBMS options would be suitable for the prototype system in terms of features as the proposed database structure is fairly simple. Due to the small scale of this project the more expensive options of Oracle Database and Microsoft SQL were eliminated, especially because both the open-source options provide all the features required. Comparing PostgreSQL and MySQL the standout differences are that PostgreSQL is fully SQL-compliant and PostgreSQL's multi-version concurrency control. This feature that will be useful when considering the "Should" requirement FS1 which would require different micro-services concurrently writing the grammar and word information from conversations to the database in real time. For these reasons PostgreSQL will be the DBMS used in the prototype system.

4.7 Docker Image Design

As previously mentioned in section 4.4 System Architecture Design, custom docker images for each of the micro-services will be created. In this section the structure and underlying software for each docker image will be discussed. It should be noted that there could be some changes during implementation due to unforeseen circumstances or implementation of any of the "Should" requirements.

4.7.1 User Interface Service

The User Interface service will be implemented using ReactJS, an open-source JavaScript framework for developing website user interfaces. ReactJS has massive popularity in industry with companies like Facebook, Netflix, Instagram and Amazon using React to develop their UIs. React's main features are the virtual DOM; this allows the selective updating of aspects of a webpage rather than updating the whole page which greatly increases speed [40], and JSX; a combination of JavaScript and HTML which is more convenient and enables the virtual DOM functionality [41]. For the prototype system,

with a previous experience using the framework and considering the time constraints, ReactJS was the framework of choice. To run ReactJS the docker image also needs to have Node.js and Node Package Manager (NPM) installed. Node.js is a JavaScript runtime used for server side development, but it is NPM, which is installed alongside Node. which is useful for ReactJS. NPM is a package manager through which a number of open-source JavaScript libraries called libraries or modules are available. NPM enables access to third-party libraries and React components provided by the React community. An example of this is the Axios library, a HTTP Client for the browser that will enable this service to send HTTP requests to the other services. NPM also manages all these libraries simplifying dependency management. To conclude this service will be a JavaScript based image with Node, NPM, ReactJS and any other third-party NPM libraries used during development installed.

4.7.2 Chatbot Service

For the chatbot service, along with housing and running the chatbot model, this micro-service also needed a REST API. This will allow the other micro-services in the prototype system to communicate with the chatbot through REST requests. Since the chatbot model provided by the Wang et al. [23] paper is written in python, this will be a python based docker image. This means a python REST API framework is the best choice to ensure easy integration and interaction. The REST API framework chosen is FastAPI, a fairly new framework gaining fast popularity and being used by Uber and Netflix. This framework has the advantages of being one of the fastest API frameworks available, automatic API documentation, build-in data type validation and dependency injection allowing unlimited application plug-ins [42]. To run the FastAPI code a HTTP server is needed and Uvicorn, a python ASGI server, is the server recommended for use with FastAPI. The full software stack of this micro-service container image is python 3.7 with the following python libraries: FastAPI; the REST API framework that will define the API for the service, Uvicorn; the ASGI server that will run the FastAPI code, Pytorch 1.4.0; an open-source machine learning library needed for the chatbot model and transformers 2.1.1; Huggingface's [43] library that provides access to an extensive set of pre-trained language models also needed for the chatbot model.

4.7.3 Grammar Extractor Service

To remain consistent across micro-services the grammar extractor service's API will also be written in python using FastAPI. Again, the docker image will run with python 3.7 and the FastAPI and Uvicorn libraries however, some additional python libraries were also included. These additional libraries are: the regex library; has more features than the python inbuilt regex library need for the grammar extraction process and the strandfordcorenlp library; handles sending requests to the coreNLP service and converts responses to a more usable structure.

4.7.4 Classifier Service

Again, to remain consistent across micro-services the classifier service's API will also be written in python using FastAPI. The docker image will run with python 3.7 and the FastAPI and Uvicorn libraries with some service specific additional python libraries. The additional libraries will be: torch, transformers and stanfordcorenlp, all libraries previously discussed.

4.7.5 User Authentication Service

As with the previous services this API was written in python using FastAPI, again with some additional python libraries. The first of these additional libraries was fastapi-jwt-auth, an open-source FastAPI extension that was used to implement the JWT functionality discussed in the security design section. The next library is Passlib that provides the hashing functionality needed for password storage, again covered in the security design section. The next library is SQLAlchemy, an object-relational-mapping (ORM) library which maps SQL database Tables to python objects, that was used to interact with the database service. The final additional library is psycopg2, the PostgreSQL database adapter for python that was used by SQLAlchemy to connect to the PostgreSQL database in the database micro-service.

4.7.6 CoreNLP Service

The coreNLP micro-service will be an instance of the Stanford CoreNLP web API server. Stanford CoreNLP provides a set of natural language analysis tools to perform tasks on raw text such as part of speech (POS) tagging, tokenization, named entity recognition or dependency parsing [44]. The CoreNLP service is written in Java meaning that the docker image was also Java based. The docker image will have Java 8 and the latest release of Stanford CoreNLP installed, but as CoreNLP defaults to English out-of-the-box, the Chinese language packs also have to be downloaded. The image built will create a Chinese instance of the CoreNLP service with tokenization and POS tagging functionality as those are the only analysis tools needed by the grammar extractor and classifier services.

4.7.7 Database Service

When implementing the database service there will be no need to create a custom docker image. The Docker community provides an extensive set of pre-made docker container images so the pre-made ‘postgres:alpine’ community image will be used instead.

4.8 API Design

After analysis the functional and non-functional requirements of the prototype system these are the projected API endpoints that will need to be implemented per service.

4.8.1 Chatbot Service API

Req ID	Endpoint	Action	Parameters	Response
FM4	/chat	Converse with the chatbot	Access token (Header) User message	Chatbot message

Table 6: Chatbot Service API Endpoints

4.8.2 Grammar Extractor Service API

Req ID	Endpoint	Action	Parameters	Response
FM6	/extract	Extract grammar patterns from user message	Access token (Header) User message	Extracted grammar patterns with their levels

Table 7: Grammar Extractor Service API Endpoints

4.8.3 Classifier Service API

Req ID	Endpoint	Action	Parameters	Response
FM5	/classify	Classify vocabulary from user message	Access token (Header) User message	Vocabulary classifications

Table 8: Classifier Service API Endpoints

4.8.4 User Authentication Service API

Req ID	Endpoint	Action	Parameters	Response
FM1	/register	Register user account	Full-name, Username, Email and password	Access token
FM2	/login	Log user into system	Username and password	Access token
N4	/refresh	Refresh access token	CSRF refresh token (Header) Refresh token cookie	Access token
FM3	/logout	Logout user from system	Access token (Header)	Status code
FS1	/conversations	Get user conversation history	Access token (Header)	Conversation history
FM4	/chat	Verify requests made to Chatbot service	User message	Response from Chatbot service
FM5, N4	/classify	Verify requests made to Classifier service	User message	Response from Classifier service
FM6, N4	/extract	Verify requests made to Grammar Extractor service	User message	Response from Grammar Extractor service

Table 9: User Authentication Service API Endpoints

4.9 User Interface

For the prototype system the decision was made to create web application to combine the different functionality of the micro-services together and present them to the user. In this section the chosen CSS framework chosen for the styling of the website is justified and UI mock ups are presented.

4.9.1 CSS Frameworks

CSS frameworks are libraries containing a collection of pre-defined and ready to use CSS style-sheets. These frameworks assist developers in quickly creating User Interfaces with cross browser support and consistent styling without having to start from scratch.

The two CSS framework considered for the styling of the prototype system were Twitter's Bootstrap [45] and Google's Material Design [46]. Both of these frameworks provide different advantages and disadvantages which are brought to light in Table 6 below.

	Bootstrap	Material Design
Advantages	Delivers a high speed of	Offers numerous components that

	development as it provides reusable blocks of code and readily available themes and templates Is ‘mobile first’ being highly responsive on multiple screen sizes. Supported by a strong community of developers, and many resources available on the official web site	provide a base design, guidelines and templates. Unlike Bootstrap does not rely on JavaScript frameworks or libraries
Disadvantages	All Bootstrap pages look the same making it difficult for developers to implement custom themes Adds unnecessary bloat to web pages generating large files and negatively impacting the loading time and battery usage of the web site	Emphasises animations which carry a lot of performance overhead and can also be distracting to users Is affiliated in Google making it difficult for developers to produce platform independent user experiences disassociate from the Google ecosystem

Table 10: Bootstrap and Material Design Evaluation

After reviewing the strength and weakness of both frameworks Bootstrap is the framework that will be chosen for the prototype system. It is the popular of the two frameworks and also fits into the User Interface ecosystem perfectly being JavaScript dependent and the service being JavaScript based.

4.9.2 User Interface Design

To aid in the design process a wireframe tool, WireFrame Pro, was used to create the application logo and different design mock-ups of webpages that will be needed by the system. These mock-ups will provide a reference when implementing the prototype system.

Figure 16: Application Logo

Figure 16 displays the logo designed for the prototype system. The Chinese characters beneath the chatbot image “中文聊天机器人” translates to “Chinese Chatbot” in English

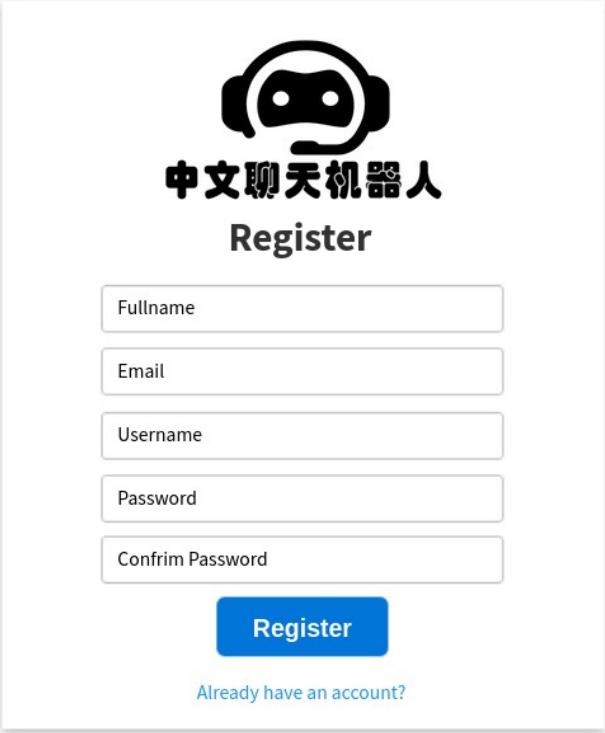


Figure 17: UI Mock-up of Registration Page

Figure 17 illustrates how the registration page could look. It will consist of the application logo and fields to hold the required user inputs. These inputs will have to be validated by the system once the register button is pressed i.e. checking that all fields are completed and the entered username is unique. A link to the login page will also be included to navigate existing users to that page to login.

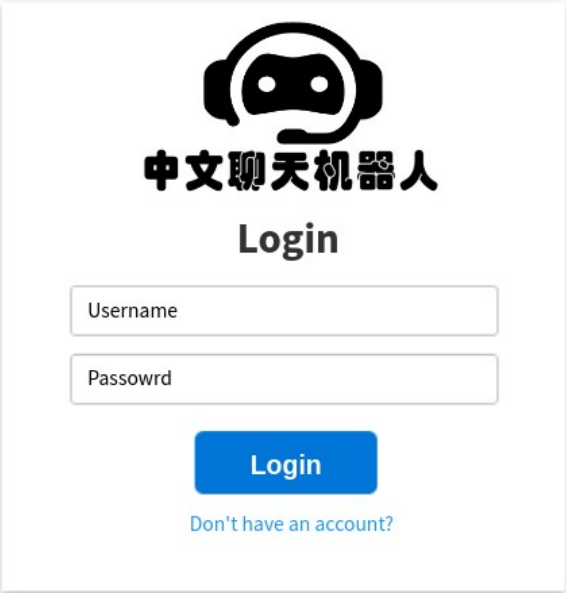


Figure 18: UI Mock-up of Login Page

Figure 18 shows that the Login page will have a similar design to the registration page, again incorporating the application logo. The input fields on this page will require a more basic form of validation i.e. just checking all the fields are completed

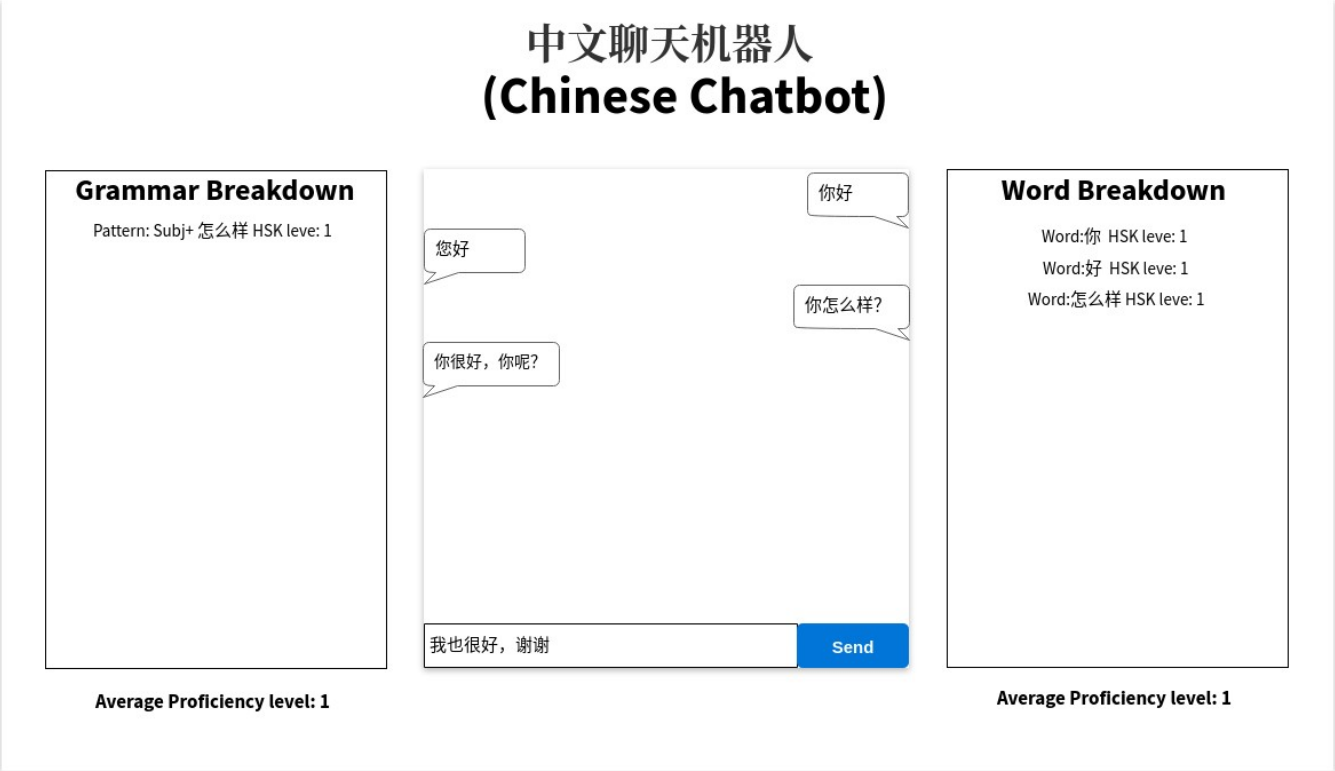


Figure 19: UI Mock-up of Chatbot Page

Figure 19 presents the possible design of the chatbot page. The centre of the page displays the chat-box containing the chat between the user and chatbot. On either side of the chat-box is the resulting analysis of the system on the words and grammar patterns used by the user during the conversation so far. Below the word and grammar information the average proficiency level of the words and grammar patterns are displayed respectively.

4.10 Summary

In this section the designs of multiple aspects of the prototype system were documented with the rational behind each decision also being highlighted. These designs will be used during the implementation of the prototype system.

5 Implementation

In this section the general code base structure is presented before an example docker file and code snippets from the docker-compose files are explained. Following this the most challenging aspects encountered during the implementation of the Chatbot, Classifier, Grammar Extractor, User Authentication and User interface services are dissected, with the coded solutions being examined. All of the code referenced throughout this section can be found in the source code submitted alongside this report.

5.1 Codebase Structure Overview

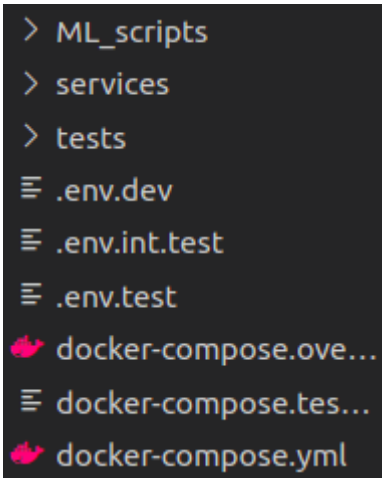


Figure 20: Codebase Main Directory

Figure 21: Codebase Service Directory

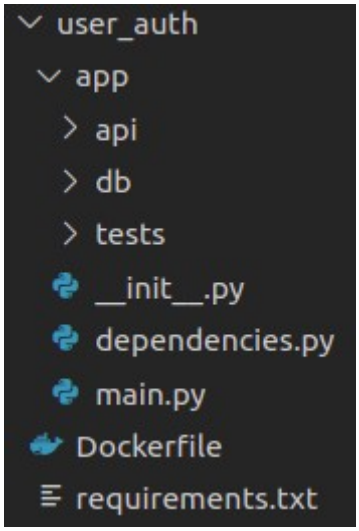


Figure 22: User Authentication Service Directory

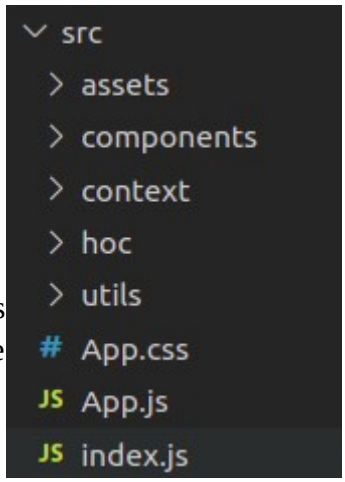


Figure 23: User Interface Service Src Directory

Figure 20 displays the structure for the main directory of the prototype system. The 'ML_scripts' directory contains the different python files and vocabulary datasets used to create the classification model, explained in further detail in section 5.4. The services directory holds the code for all the different services of the system observable in Figure 21. The tests directory holds the intergration_test.py file used for integration testing; a topic covered in section 6.2. The different '.env' and 'docker-compose files' are also both discussed in the following sections. Figure 22 illustrates the directory structure of the User Authentication service ('user_auth'). The first level of the directory holds the Dockerfile and requirements.txt files used to create the docker container image of this service. The app subdirectory contains the source code for the service. The 'api' directory consists of helper functions and configurations used by the service, the 'db' directory contains sqlalchemy specific code used by the service to connect to the database service. The tests directory holds the unit tests for the service discussed in section 6 Testing, dependencies.py contains the FastAPI dependencies clarified in section 5.7 and main.py consists of the FastAPI code defining the services endpoints. All the services built using FastAPI, the Classifier, Grammar Extractor, Chatbot and User Authentication services, all have a similar directory structure with the 'db' directory and dependencies.py being removed depending on if the service

communicates with the Database service and the ‘api’ directory being replaced with a service specific directory filled with service specific code. Finally, Figure 23 displays the ‘src’ folder found in the directory for the User Interface service (‘ui’). This directory contains all the ReactJS code that compiles to create the user interface of the system. The ‘assets’ directory contains all of the different images used in the application. The ‘components’ directory contains the different ReactJS components created for the user interface. The ‘context’ directory contains the authentication context, a concept explained in section 5.6. The hoc directory contains the authentication route higher-order component , another concept explained in section 5.6. The ‘utils’ directory contains the utility functions used throughout the application such as the API client, used to send request to the User Authentication Service and the in memory JWT storage implementation. Finally, App.js and index.js loads the different resources and combines the ReactJS components producing the final user interface.

5.2 Docker

5.2.1 Example Docker File

```
FROM tiangolo/uvicorn-gunicorn-fastapi:python3.7

RUN mkdir /user_auth

COPY requirements.txt /user_auth

WORKDIR /user_auth

RUN pip --no-cache-dir install -r requirements.txt

COPY . /user_auth

EXPOSE 8005

WORKDIR /user_auth/app
```

Figure 24: User Authentication Service Docker File

Figure 24 displays the Dockerfile which produces the user authentication docker image in the prototype system. The Docker community provides an extensive set of pre-made docker container images and the official FastAPI image, tiangolo/uvicorn-gunicorn-fastapi:python3.7, was used as the base image for this micro-service container image. This is signified in the first ‘FROM’ statement in the first line of the file. It should be noted that this community docker image also includes an additional python library Gunicorn; a WSGI HTTP Server that can be combined with Uvicorn to create an asynchronous multi-process server for deployment. Next a ‘user_auth’ directory is created in the docker image and the requirements.txt file is copied from the host machine into this directory. This requirements.txt file contains the other python libraries needed for this image previously introduced in section 4.7: SQLAlchemy, psycopg2, fastapi-jwt-auth, passlib[bcrypt]. Two additional python libraries were also added during development of the system: httpx; a python HTTP client whose use will be covered in section 5.7 and the wait-for-it whose use case will be covered in section 5.2.2. These libraries are

installed in the docker image before the code that will be run on the image is copied over from the host machine. Finally, the port 8005 is exposed to enable the service's API to receive HTTP requests.

5.2.2 Docker Compose Files

As previously stated in section 4.4 System architecture design, Docker-compose was used to define the network of docker containers in the prototype system, allowing the different service to communicate. This is defined using a docker-compose.yml file which consists of information such as the different services that will be included in the network, and the type of network written in YAML. YAML stands for "YAML Ain't Markup Language" and is a human-readable data serialization standard often used to write configuration files [47]. For the prototype system multiple docker-compose files were used to facilitate separate test and develop environments. These three files in the code base are docker-compose.yml, docker-compose.override.yml and docker-compose.test.yml. The docker-compose.yml file contains the service definitions that are shared between the two environments, docker-compose.override.yml contains the development specific services and configurations and docker-compose.test.yml contains the testing specific configurations. There are also three different .env files that contain the different environment variables needed for the different environments. env.dev contains the environment variables for the development environment, env.test contains the environment variables used in the test environment and env.int.test contains the environment variables for the integration testing environment which are slightly different. The testing environments are explored further in section 6.

```
classifier:
  build: classifier/
  environment:
    ENV: "${ENV}"
    POSTGRES_DB: "${POSTGRES_DB}"
    POSTGRES_USER: "${POSTGRES_USER}"
    POSTGRES_PASSWORD: "${POSTGRES_PASSWORD}"
    POSTGRES_PORT: "${POSTGRES_PORT}"
  networks:
    - fyp_network
  container_name: classifier
```

Figure 25: Docker Compose Classifier Service Definition

```
networks:
  fyp_network:
    driver: bridge
```

Figure 26: Docker Compose Network Definition

The docker-compose.yml file contains the service definitions for the chatbot, classifier, grammar extractor and user authentication services and the network definition. Figure 25 and 26 are code snippets taken this file. Figure 25 highlights an example of a service definition, defining the classifier service of the system and Figure 26 is an example of defining a docker network. In Figure 26 the docker network of the system as 'fyp_network' that uses the bridge driver to facilitate communication between services is defined. In Figure 25 'build' points to the directory location of the docker file that builds the container image for this service, 'environment' maps the environment variables from the specified .env file to the environments variables of the container, 'networks' adds the service to the fyp_network specified in Figure 25, and 'container_name' sets the name the other services in the network can use as a reference to communicate with this service.

```
classifier:
  volumes:
    - './classifier/app:/classifier/app'
  ports:
    - 8001:8001
  command:  bash -c "wait-for-it
    -t 30
    -s db:${POSTGRES_PORT}
    -s corenlp:9001
    -- uvicorn main:app --host 0.0.0.0 --port 8001 --reload "
```

Figure 27: Docker Compose Classifier Service Development Configuration

Docker-compose manages multiple docker-compose files through merging the configurations of the services that are defined across the files. Figure 27 displays the development configurations for the classifier service from the docker-compose.override.yml. These configurations are merged with the classifier service definition from the docker-compose.yml file presented earlier in Figure 25.' volume' maps the designated directory on the host system, './classifier/app', to a directory in the docker container , /classifier/app. The addition of this volume to the service configuration facilitates reloading the docker container after any changes are made to the service's source code during development. 'ports' maps the ports of the docker container to the ports of the host machine and 'command' specifies the command to be run when the service is started. For the Chatbot, Classifier and Grammar Extractor services in the network the command executed uses the python library wait-for-it to delay the start up of the container. This delay occurs until other services in the network that the service depends on have started running successfully. The command in Figure 27 waits for the Database and coreNLP services to be running before starting the uvicorn service on port 8001 with '--reload' flag to restart the server on code changes. The User Interface service is also started with a similar command but uses the original wait-for-it bash script that the python library is based on [48].

```
db:
  image: postgres:alpine
  restart: always
  environment:
    POSTGRES_DB: "${POSTGRES_DB}"
    POSTGRES_USER: "${POSTGRES_USER}"
    POSTGRES_PASSWORD: "${POSTGRES_PASSWORD}"
  ports:
    - "${POSTGRES_PORT}:${POSTGRES_PORT}"
  volumes:
    - ./db/${ENV}:/docker-entrypoint-initdb.d/
  networks:
    - fyp_network
  container_name: db
```

Figure 28: Docker Compose Database Service Definition

The docker-compose.override.yml also denotes the service definitions for the services only needed by the development environment; the Database, coreNLP and User Interface services. Figure 28 is the service definition of the Database service which differs slightly to the classifier service definition shown in Figure 25. In this example ‘image’ replaces ‘build’ as this service uses a community image instead of a docker file to build the docker container. ‘restart’ ensure this container is always restarted if it stops running due to an error. ‘volumes’ moves different .sql files to the database based on the ‘ENV’ environmental variable allowing the database to be filled with different dummy data depending on the environment being run.

Finally to run the prototype system in development mode using Docker-compose, the command ‘**docker-compose --env-file ./.env.dev up -d**’ is executed in the directory containing the docker compose files. This command automatically finds the docker-compose.yml and docker-compose.override.yml files, merges their configurations, and starts the prototype system.

5.3 Chatbot Service

To implement this service the most important consideration was the implementation of the generative chatbot and for this purpose the work of Wang et al. [23] was utilised. Their CDialGPT LCCC-large model was fine-tuned on the Short-Text-Conversations (STC) dataset, using the train.py script provided from the git repo associated with their paper [49]. The STC dataset ,provided by Shang et al. Paper [50], contains roughly 4.4 million pairs of conversation from Weibo, the Chinese equivalent to Twitter. As per the Wang et al. paper, the model was fine-tuned for 10 epochs with the AdamW optimizer, a linear learning rate decay method, a learning rate of 6.25e-5, a batch size of 8 and a gradient accumulation of 64. The fine-tuning process was completed using HTCondor [51] to run jobs on three Nvidia Quadro RTX 6000 GPUs in the university’s GPU pool [52]. The resulting model of the fine-tuning process is the chatbot model users communicate with through the prototype system. This chatbot model is accessed via the web-socket endpoint ‘/chat’ on the Chatbot service’s API. A web-socket connection provides a bidirectional, full-duplex communications channel which accommodates the chatbot use case as a connection can be established with the chatbot and messages exchanged in real time. Each web-socket connection has its own dedicated instance of a chatbot due to the message history of the conversation being used by the chatbot when generating responses.

The main challenge that had to be overcome during the implementation of this service was the chatbot model on occasion continuously producing the same output response to different user messages. Referring to the code base of the prototype system within the ‘services/chatbot/app/chatbot_utils’ directory there is a chatbot.py file containing the ‘Chatbot’ python class used to create instances of the chatbot model. To rectify this issue, a loop and conditional were inserted into the ‘chat’ function within the class exhibited in Figure 29 below. This checks that the response generated is not identical to any of the other chatbot responses returned in the conversation so far.

```
while True:
    out_ids = self.generate_out_ids()
    if out_ids not in self.history[index::2]:
        break
```

Figure 29: Chatbot Chat Function Loop and Conditional Fix

The While loop uses the ‘generate_out_ids’ function to produce ‘out_ids’, the tokenized response of the chatbot, until a response that doesn’t previously exist in the history list is produced. This loop and

conditional prevents the chatbot generating identical responses at the character level, but does not prevent the chatbot producing responses that are semantically identical. To remedy this new issue a means to determine the level of semantic similarity between two sentences had to be devised. To do so two additional python libraries: Gensim; a NLP python library used for topic modelling, document indexing and similarity retrieval and Scipy; a python library that specializes in number calculations, were employed. Using Gensim the fasttext Chinese pre-trained word2vec model [53] was loaded and used to convert the words in each sentence into vectors, before calculating the average vector of the sentence. Next Scipy's 'cosine' function was applied to calculate the cosine similarity between the two sentence vectors, resulting in a similarity score for the two sentences. Combining another conditional with this sentence similarity calculation in the 'chat' function, if a certain similarity threshold was breached the generated chatbot response would be replaced by a random question, chosen from a set of predefined questions in questions.txt. These questions were extracted from the ChatterBot Corpus [54], a set of dialogue data constructed for use with the ChatterBot library; a python library used to produce retrieval-based chatbots [55]. This sentence similarity conditional is outlined in Figure 30

```
prev_comparison=compare_to_history(self.history[-2], possible_out_text, self.tokenizer)
if prev_comparison['similarities'][0] > 0.95:
    history, new_out_text=get_random_question(self.history,self.tokenize)
    self.history = history
    out_text=new_out_text
else:
    history_comparisons=compare_to_history(self.history[index:-2:2], possible_out_text, self.tokenizer)
    if history_comparisons and history_comparisons['similarities']:
        if any(similarity > 0.95 for similarity in history_comparisons['similarities']):
            print(history_comparisons)
            history, new_out_text=get_random_question(self.history,self.tokenize)
            self.history = history
            out_text = new_out_text
        else:
            self.history.append(out_ids)
            out_text =  possible_out_text.replace(" ", "")
    else:
        self.history.append(out_ids)
        out_text =  possible_out_text.replace(" ", "")
```

Figure 30: Chatbot Sentence Similarity Conditional

The first call to the 'compare_to_history' function compares the 'possible_out_text', the 'out_ids' converted to text, to the last message sent by the chatbot and stores the results in the 'prev_comparison' variable. If the sentence similarity is over 0.95 then the 'out_text', the response returned to the user, is assigned a random question. If the sentence similarity is less than 0.95 then the else clause is triggered and a second call to the 'compare_to_history' function is made to find the similarity between the 'possible_out_text' and all the other messages sent by the chatbot. Again, if the sentence similarity between the 'possible_out_text 'and any of the chatbot's previous response is greater than 0.95 the 'out_text' assigned a random question. If none of the conditionals are triggered, then the original 'possible_out_text' is assigned to 'out_text' and sent as a response to the user. This method ensures that the chatbot does not produce identical response in any form however at times the random question returned may not fit the previous topic of conversation. The semantic similarity functionality can be found in the sentenceSimialirty.py file in the form of the 'compare_to_history', ' compare_sentences' and 'get_random_question' functions.

The new similarity functionality however, introduced a different issue as the size of the fastttext word2vec model resulted in an oversized docker image. To resolve this the dimensions of the model were reduced from 300 to 100, and just the word vectors were extracted and saved with the model being discarded. This process was carried out by the preprocessingModel.py script that is called during the build process of the container image, and greatly reduced the storage requirements for the vectors.

5.4 Classifier service

The most crucial step of implementing the classifier service was creating the classification model. Multiple experiments were conducted to determine the best dataset, model and hyper-parameters to produce the most accurate classification model possible.

From the CLUE leader-board three different pre-trained Chinese models were chosen: Bert base, Albert-tiny and Roberta large, with both Albert and Roberta being variations of the original Bert model. In comparison to Bert, Roberta: is trained on a significantly larger dataset; 160GB vs Bert's 16GB, incorporates a dynamic masking pattern instead of a static masking pattern, replaces the next sentence prediction (NSP) objective with an objective of full sentences without NSP and can be trained on longer sequences [56]. Albert focuses on decreasing the size of the original Bert model while still maintaining similar performance. This is achieved as a result of adopting Embedding Matrix Factorization and Cross-layer parameter sharing to reduce the size of the input parameters by 89%, and replacing the NSP objective with its own sentence order prediction objective [57]. Each of these models had an additional linear layer added for the task of classification. Access to all these models were provided by the Huggingface transformers python library.

The HSK and TOCFL vocabulary lists were adapted into three datasets: a HSK dataset, a TOCFL and a HSK_TOCFL dataset; a combination of the two lists. The HSK_TOFCL and TOCFL datasets were split into separate train and test datasets using a 80:20 train test split so that the model could be evaluated on an unseen dataset at the end of training. The HSK dataset was also split but using a 90:10 train test split due to it containing approximately 2000 less data points than the other datasets. These datasets can be found in the 'ML_scripts' directory of the codebase.

The train.py script, also in the 'ML_scripts' directory, performs the training process on each model for 4 epochs four times with the different training datasets and a 90:10 train validate split. Before training, each dataset had to be pre-processed into a form that the models could understand. Utilizing the respective model's tokenizer each word was converted into a combination of input_ids and attention masks, which are the values that were passed to the models during training. A batch size of 64, learning rate of 1e-4, the AdamW optimizer and the cross-entropy loss function were the hyper-parameters set during training for all the models. The 1 cycle learning rate policy from the Smith et al paper [58] was chosen as the learning rate scheduler for training. This scheduler uses only one cycle with 2 steps of equal length and a cycle length slightly less than the total number of training epochs. A cycle is the number of iterations where the learning rate goes from its lower bound to its upper bound and then back to its lower bound and a step is half of a cycle. The learning rate is taken from its initial value to some maximum learning rate during the first step before being taken back to the initial learning rate during the second step. For the remaining iterations, the learning rate is then annihilated to a value significantly lower than the initial learning rate. The benefits of this policy are that it allows complicated models to increase the speed of convergence and the high learning rate during the middle training acts as a regularization method preventing the model from over-fitting. A problem occurring across all the three datasets is that the classes are extremely unbalanced owing to the general nature of

vocabulary lists; higher proficiency levels have more vocabulary. To try and alleviate this oversampling was implemented in the form of a weighted random sampler. This was deployed as the training data-loader to ensure that an equal proportion of all the class were in each training batch.

```
for step, batch in enumerate(train_dataloader):
    batch_counts +=1
    inputs, labels = tuple(move_to(t,device) for t in batch)

    model.zero_grad()
    outputs = model(inputs)
    loss = criterion(outputs, labels)
    batch_loss += loss.item()
    total_loss += loss.item()

    preds = torch.argmax(outputs, dim=1).flatten()

    accuracy = (preds == labels).cpu().numpy().mean() * 100
    train_accuracy.append(accuracy)

    loss.backward()

    torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

    optimizer.step()
    scheduler.step()
```

Figure 31: Classification Model Training loop

Figure 31 displays the training loop used to train the models. This loops through all the batches of training data in the 'train_dataloader' and for each batch: moves the training data to the GPU, clears any previously calculated gradients and performs a forward pass; passes the inputs to the model. The outputs from the model are then used to complete multiple process. First, the loss is calculated before being added to the 'batch_loss' and 'total_loss' which are used to calculate the average batch and epoch loss. Next the output is converted to predictions and used to calculate the training accuracy of the model. Next the loss is used to perform a backward pass to calculate the model gradients, the gradients are clipped to prevent the 'exploding gradient' problem, the parameter are updated and a step is taken using the computed gradient, dictated by the scheduler. Finally, the learning rate is updated. Validation was performed at the end of every training epoch to gauge which of the models produces the best accuracy on which dataset and the average of the results for these experiment can be observed below in Table 11.

Model	HSK Dataset Validation Accuracy	TOCFL Dataset Validation Accuracy	HSK_TOFCL Dataset Validation Accuracy
Bert-base-chinese	56.64	44.80	43.79
Albert_tiny	51.25	40.14	31.00
Roberta_large	58.04	45.09	45.96

Table 11: Average Model Validation Accuracies on the Different Proficiency Test Vocabulary Datasets

Across all the datasets, Albert had the worst performance with the lowest validation accuracies while Roberta performed the best. All the models also achieved their greatest validation accuracies being trained on the HSK dataset. This is most likely due to this dataset only having 6 classification classes, HSK has 6 levels of proficiency, while TOCFL and HSK_TOFCL have 7. Considering the results, the Roberta model with the HSK dataset would seem like the most logical choice for development of the final classification model. However, there was another factor that was also considered, the model size. Roberta is double the size of Bert requiring 1.3 GB of storage, but only produces a 2% improvement in validation accuracy on the HSK dataset. Taking into consideration the performance of the entire system, using Bert with the HSK dataset was the better choice as it required less memory whilst providing similar results.

A separate hyperparam_optimizer.py script was written that harnesses the ray-tune and wandb python libraries for hyper-parameter optimization of the Bert classification model. 100 different models were trained using a set of learning rates ranging from 1e-5 to 1e-4 to find the learning rate that led to the best validation accuracy. The validation accuracy of the top 5 models is exhibited below in Figure 32.

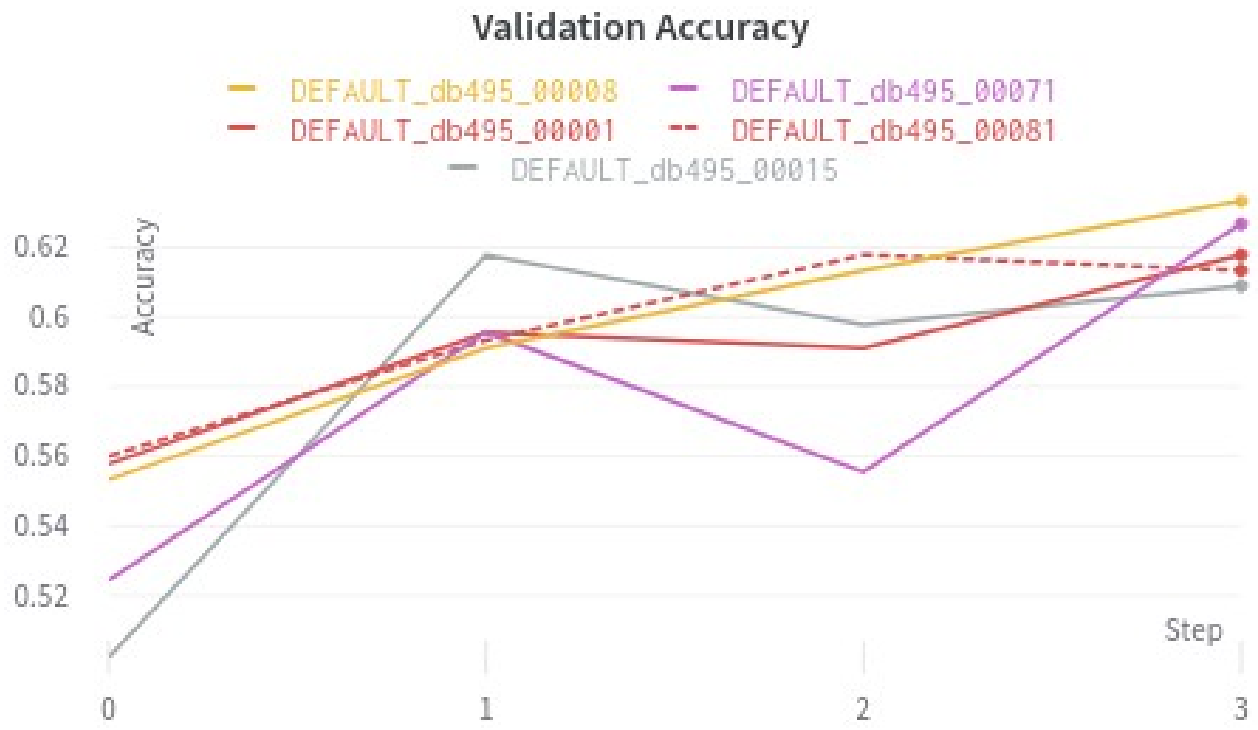


Figure 32: Validation Accuracy of Top 5 Trial Models

The DEFAULT_db496_00008 model represented by the golden line in the graph produced the best validation accuracy of 63.33% with a learning rate set to 3.738e-5.This learning rate was used to produce the final model.

Finally, the train.py script was again employed to train the final model integrated in the prototype system. During this training process the same number of epochs, batch size optimizer, loss function and scheduler as the first experiment were set. However, no validation at the end of each process was carried out and the optimal learning rate discovered during hyper-parameter optimization was assigned. The subsequent model was then tested using the unseen HSK test dataset. Following multiple attempts, the highest accuracy on the test set achieved was 60%. This is far from the desired accuracy but considering that this is a prototype system and the time constraints of the project, the decision was made to stop experimentation and continue development with the 60% accuracy model.

This model is wrapped in the 'WordLevelClassifier' python class defined in the word_level_classifier.py file. This file is found in the 'services/classifier/app/classification_model' directory. This class is comprised of several helper functions used to interact with the classification model. An instance of this class is initiated on the start up of the service's server, and is used to complete Classification requests made to the '/classify' endpoint of the the service's API.

It should also be noted that for the implementation of the FS1 requirement the sqlalchemy and psycopg2 python libraries were added to the service to facilitate access to the database service.

5.5 Grammar Extractor service

Allset Learning is a Shanghai-based learning consultancy dedicated to helping foreigners better learn Mandarin Chinese [59]. Their online grammar wiki provides a comprehensive set of Chinese grammar rules associated to CEF and HSK levels of proficiency. It is these rules that form the basis of the grammar extraction functionality. To implement the grammar extractor service a method for extracting these grammar rules from a Chinese sentence, and also associating these patterns to their level of proficiency had to be created. This task in it's self was challenging as it was not clear what would be the best approach to take.

The first method explored was chunking. In natural language processing (NLP), chunking is the technique of extracting phrases, known as chunks, from text based on the Part of Speech (POS) tags of the words in the text. POS tags signify a words grammatical position in a sentence i.e. noun, verb, adjective, based on its definition and context. A chunking rule for a noun phrase chunk can be defined as a phrase that has zero or one determiner, followed by zero or more adjectives and then a noun. If this this chunking rule was applied to the sentence 'The brown dog runs fast', the chunk extracted would be 'The brown dog'. This is because the word 'The' is a determiner 'brown' is an adjective and 'dog' is a noun. This method was not suitable for the grammar extractor use case as the majority of the Allset grammar rules are based around a particular Chinese character i.e. 都 + Verb / Adj. It is not possible to include actual words or characters in the definition of chunking rules, only strictly POS tags. The next method that was investigated was context free grammars. Context-free grammars can be described as a set of rules that define a language but are not influenced by context. They are made up of terminals; a set of words or characters that make up the actual content of the language, non-terminals; a sub language within the language defined by the grammar that act as placeholders for terminals, the start symbol; a special non-terminal that represents the initial string and production rules; the set of rules that define the grammar. Context free grammar did support the format of the Allset grammar rules but

presented a few different complications. Firstly, as the entire rule set would have been defined in a single large context free grammar, it would be possible to determine if a sentence used a rule but not which rule. Secondly, if the rule used could not be determine neither could the grammatical level of the sentence.

The final method and solution involves using regex patterns and the Stanford Core NLP's POS tags. The grammar extractor service first takes a sentence and sends it to the coreNLP service where it is tokenized, annotated with POS tags and is returned as a list of values. An example response is showcased below.

[(' 你 ', 'PN'), (' 好 ', 'VA'), (' ， ', 'PU'), (' 你 ', 'PN'), (' 怎么样 ', 'VA'), (' ？ ', 'PU')]

Figure 33: coreNLP Service Example Response

This list is then concatenated together to form a structure similar the original sentence but with each word being followed by their relevant POS tag. This structure has been named the POS sentence and an example POS sentence, formed from the list in Figure 33, can be examined in Figure 34.

' 你 PN 好 VA ， PU 你 PN 怎么样 VA ？ PU'

Figure 34: Grammar Extractor Example POS Sentence

Predicated on this POS sentence structure and alongside the Penn Chinese Treebank POS tagging guidelines [60], a collection of regex patterns were created that reflect the Allset grammar rules. These regex patterns facilitate the extraction of different grammar rules from a target sentence. It should be noted that as a result of the time constraints only the first 100 of the Allset grammar rules were implemented

{'word':' 太 ','rules':' 太 + Adj. + 了 ','pattern':fr' 太 AD{adj} 了 (?:AS|SP)', 'level':1}

Figure 35: Grammar Dictionary One Pattern

{'word':' 也 ','rules': [{'rule':'Subj. + 也 + Verb / [Verb Phrase]',
'pattern':fr'{subj}AD{verbPhrase}', 'level':1},
{'rule':'Subj. + 也 (+ Adv.) + Adj',
'pattern':fr'{subj} 也 AD{adverb}*{adj}', 'level':1}]}

Figure 36: Grammar Dictionary Multiple Patterns

Each regex pattern was placed in a dictionary as can be seen in Figures 35 and 36. These 'Grammar' dictionaries have four keys: word; the word relating to the grammar rule, rules; the rule definition, pattern; the regex pattern to extract the rule and level; the level of the rule. In some cases, the same word had multiple rules at different levels, in which case the rules key would instead store a list of dictionaries, each with their own rule definition, regex pattern and level exemplified in Figure 36. The grammar extraction process entails looping through a list of these grammar dictionaries, checking for the presence of a word associated with a rule in the input sentence and, if found, applying the relating regex pattern or patterns. A successful phrase extraction using a regex pattern signifies that the corresponding grammar rule was used in the sentence. This grammar extraction functionality is implemented in the 'parseSentence' and 'extractGrammar' functions defined in the grammarUtils.py file.

```python
def parseSentence(sentence):
    tags = parser.pos_tag(sentence)
    pos_sent = ''.join([i for sub in tags for i in sub])
    return pos_sent
```

Figure 37: Grammar Extractor Parse Sentence Function

The ‘parseSentence’ function in Figure 37 handles the creation of the POS sentences as displayed in the Figure 34 above. First, the function makes a request to the coreNLP service and receives a list of tuples as a response. Next, it loops through this list extracting each the word and POS tag from each tuple, storing them in a separate list. Finally, the words and POS tags are joined together as a string; the POS sentence and returned by the function.

```python
def extractGrammar(sentence):
    grammarExtracted=[]
    possibleMistakes=[]
    pos_sent= parseSentence(sentence)
    for grammar in grammarRules:
        if re.search(grammar['word'],sentence):
            if type(grammar['rules']) is str:
                p=re.compile(grammar['pattern'])
                matches=p.findall(pos_sent)
                if len(matches)==0:
                    possibleMistakes.append(grammar['word'])
                else:
                    grammarExtracted.append({'rule':grammar['rules'],'level':grammar['level']})
            else:
                numofMatches=0
                for pattern in grammar['rules']:
                    p=re.compile(pattern['pattern'])
                    matches=p.findall(pos_sent)
                    if len(matches)>0:
                        numofMatches+=1
                        grammarExtracted.append({'rule':pattern['rule'],'level':pattern['level']})
                if(numofMatches==0):
                    possibleMistakes.append(grammar['word'])
    return grammarExtracted, possibleMistakes
```

Figure 38: Grammar Extractor Extract Grammar Function

The ‘extractGrammar’ function performs the grammar extraction process, looping through the ‘grammarRules’ list defined in grammarRules.py. For each grammar rule, first the existence of the grammar rule word in the sentence is checked. If the word does exist in the sentence, then whether the grammar rule just has a single rule is checked. If so, that regex pattern for that rule is applied to the sentence and if there is a match the rule is added to the ‘grammarExtracted’ list, otherwise it is added to the ‘possibleMistakes’ list. If the grammar rule has multiple rules then those rules are looped through and the same process of applying regex patterns and adding the pattern to the ‘grammarExtracted’ for ‘possibleMistakes’ lists occurs. The ‘possibleMistakes’ lists could contribute to the implementation of the FS5 requirement in future iterations of the system but for the current prototype system was left out.

5.6 User Interface Service

Regarding this service, the most challenging aspect to develop was the website behaviour surrounding user authentication detailed in section 4.5 Security Design. To store the JWT token in memory an implementation presented in Janvier's article was utilised [61] in the inMemoryJWT.js file, with the rest of the functionality being implemented via two advanced ReactJS features, Context and Higher-Order Component (HOC).

5.6.1 Context

In ReactJS, data is passed between components via props in a top down, parent to child fashion. However, there are times where this can be a hindrance for certain types of props, in the case of this system the authentication functions, which are required by the majority of components in an application. The ReactJS context provides a way to share this data between components without having to explicitly pass them to each component as props [57]. The Auth.js file holds the code for the authentication context used in this application. This provides all components access to the 'logout' and 'checkAuth' functions.

```
const logout = () => {
  deleteLogout().then(() => {
    inMemoryJWT.eraseToken()
    history.push("/")
  }).catch(error => {
    console.error(error)
  })
}
```

Figure 39: User Interface Logout Function

The 'logout' function uses the 'deleteLogout' function defined in api-client.js to send a delete request to the User Authentication service. This request unsets the refresh_token_cookie so a new access token cannot be requested from the service, thus signifying the end of a user session. Once this request is completed successfully the in memory JWT token is erased and the user is redirected back to the login page.

```
const checkAuth = () => {
  if (inMemoryJWT.getToken() === null && Cookies.get('csrf_refresh_token')) {
    inMemoryJWT.getRefreshedToken()
  }
  return inMemoryJWT.waitForTokenRefresh().then(() => {
    if (inMemoryJWT.getToken() === null) {
      return Promise.reject()
    } {
      let tokenExp = jwt_decode(inMemoryJWT.getToken()).exp
      let current_time = Date.now() / 1000;
      return current_time >= tokenExp ? Promise.reject() : Promise.resolve();
    }
  });
}
```

Figure 40: User Interface Check Auth Function

The ‘checkAuth’ function is responsible for assessing whether a user is still authorised to have access to the system. The first conditional checks for the existence of the ‘csrf_refresh_token’ cookie with no JWT token stored in memory, suggesting that a silent fresh should occur. This function then returns a Promise which is the main technique for handling asynchronous tasks in JavaScript. A Promise has 3 states: pending; the task is being performed, resolved; the task was completed successfully and rejected; the task failed [62]. The ‘waitForTokenRefresh’ function is a promise that makes a request to the User Authentication service for a new access token. The ‘.then’ contains the code that will be run if the promise completes with the state resolved. If there is no new access token after the ‘waitForTokenRefresh’ function call, ‘checkAuth’ returns a promise reject object. If that initial conditional is not triggered another check is performed to ensure that the stored access token is not expired. If it is, again a promise reject object is returned otherwise a promise resolve object is returned. An example execution of this ‘checkAuth’ function will be covered in the next section 5.6.2.

```
return (
  <AuthContext.Provider value={{logout, checkAuth }}>
    {children}
  </AuthContext.Provider>
);
```

Figure 41: User Interface Auth Context Provider

```
<AuthProvider >
  <App />
</AuthProvider >
```

Figure 42: Auth Provider Wrapping Application

Finally, for the components to have access to these functions a context provider is created. This allows components that subscribe to the context to have access to changes in the context [63]. Figure 41 demonstrates the definition of the authentication context provider with the value prop containing the ‘logout’ and ‘checkAuth’ functions that will be made available through the context. To subscribe all the components in the application to the context, giving them all access to the ‘logout’ and ‘checkAuth’ functions, this context provider is wrapped around the App component as shown in Figure 42.

5.6.2 Higher-Order Component (HOC)

An HOC takes another component as its input and transforms it into a new component. This allows additional functionality to be added to a component without having to directly edit the original component itself [59]. This is useful when applying the same functionality across multiple components, or working with third-party components. In this application the HOS is the authentication route whose definition can be found in the AuthRoute.js file. This component wraps around the Route component from the react-router-dom library, which is responsible for the client-side routing performed by the application. The AuthRoute HOC adds authentication to the routes, checking if the user is authenticated before giving them access to the page or redirecting back to the login page. To perform the authentication the ‘checkAuth’ function from the Auth context is imported and utilised in conjunction with the React ‘useEffect’ and ‘useState’ hooks.

```
const [authState, setAuthState] = useState('loading');
useEffect(() => {
  checkAuth().then(() => {
    setAuthState('loggedin')
  }).catch(() => {
    setAuthState('redirect')
  });
});
```

Figure 43: Auth Route Use State and Use Effect React Hooks

A React hook is simply a function that provides access to certain React features without having to create a React class [60]. The ‘useState’ hook gives access to React state, with each competent having its own local state where it stores values that belong to it. If any of these values in the state are changed then the component re-renders. As highlighted in Figure 43 the value ‘authState’ is added to the HOC’s local state with an initial value of loading with the ‘setAuthState’ function being defined to permit this value to be updated. The ‘useEffect’ hook provides access to the component’s lifecycle. Each React component has a lifecycle made up of three phases, mounting, updating and unmounting, which can be monitored and also manipulated [64]. Utilising the ‘useEffect’ hook, every time the HOC is rendered the ‘checkAuth’ function is called. If the ‘checkAuth’ function returns a promise resolve object then the code in the ‘.then’ is triggered and the ‘authState’ is changed to ‘loggedin’. If the function returns a promise reject object then the code in the ‘.catch’ is triggered and the ‘authState’ is changed to ‘redirect’. Using the value of ‘authState’ the ‘AuthRoute’ then knows what page to display to the user.

```
const requireAuthRoutes = ['ConversationHistory', 'Home', 'Chat']
const noAuthRoutes = ['Login', 'Register']

if (authState === 'loading') {
  return (
    <div className='home'>
      <Spinner type="border" color="primary" size="20" />
    </div>
  )
}
return (
  <Route
    {...rest}
    render={props => {
      if (authState === 'loggedin') {
        if (requireAuthRoutes.includes(Component.name)) {
          return <Component {...props} />
        } else {
          return <Redirect to="/home" />
        }

      } else {
        if (noAuthRoutes.includes(Component.name)) {
          return <Component {...props} />
        } else {
          return <Redirect to="/" />
        }
```

Figure 44: Auth Route Render Conditionals

If the ‘authState’ is ‘loading’ a loading spinner is presented to the user as exhibited in Figure 44. If the ‘authState’ is ‘loggedin’ then the name of the component to be displayed is checked against a list of components that require authentication; the ‘requireAuthRoutes’ variable. If the component name exists in that list, the page is displayed otherwise the user is redirected to the home page. This redirection to the home page prevents the user navigating back to the login or registration page once they have already logged in. If the ‘authState’ is anything else i.e. ‘redirect’ then the name of components is checked against the components that do not need authentication; the ‘noAuthRoutes’ variable. If the page does not require authentication it is displayed otherwise, the user is redirected to the login page. This prevents the user having access to the authenticated routes when they are not logged in or their access token has expired.

5.7 User Authentication Service

The User Authentication service’s API has multiple endpoints which act as gateways to the core services in the system. The most difficult aspect of this service to implement was the gateway like functionality. This service uses the POST ‘/classify’, POST ‘/extract’ and web-socket‘/chat’ endpoints to authenticate and forward requests made to the Classifier, Grammar Extractor and Chatbot services respectively. These endpoints are also all examples of protected endpoints which expect to receive a valid access token with every request.

```
@app.post('/classify')
async def classify(message: schemas.ChatMessage, response: Response,
                   db: Session = Depends(get_db),
                   Authorize: AuthJWT = Depends()):

    Authorize.jwt_required()
    current_user = Authorize.get_jwt_subject()
    conversation = crud.get_active_conversation(db, username=current_user)
    if conversation:
        url = f'http://classifier:8001/classify?conversationId={conversation.conversation_id}'
        async with AsyncClient() as client:
            proxy = await client.post(url, json=message.dict())
        response.body = proxy.content
        response.status_code = proxy.status_code
        return response
    else:
        raise HTTPException(
            status_code=404, detail="No active converstaion")
```

Figure 45: Classify Endpoint Function

Figure 45 demonstrates the implementation of the classify endpoint using FastAPI. The decorator on the first line denotes the endpoint path and HTTP method, in this case ‘/classify’ and POST. The endpoint function’s input parameters represent the different variables the function needs, the first being ‘message’. This variable maps the body of the received request to a previously defined Pydantic model, in this instance ChatMessage defined in schemas.py. The mapping to this model performs validation on the request body and generates a JSON schema necessary for OpenAPI schema. The second variable is the FastAPI ‘response’ object which returns the response from the classifier service. The next two variable ‘db’ and ‘Authorize’ are examples of dependency injection. Dependency Injection is a way to declare external code that the endpoint requires and FastAPI takes care of providing those

dependencies. The ‘db’ variable is an SqlAlchemy Session and depends on the ‘get_db’ function to retrieve a session connection to the database in stored in the database service. ‘Authorize’ is the dependency needed to integrate with the fastapi-jwt-auth library. Leveraging the ‘Authorize’ variable the endpoint checks that the received request has a valid access token, if not an HTTP 422 error response is sent. If so, utilising the ‘db’ variable, the ‘get_active_conversation’ function defined in crud.py and the username of the current logged extracted from the access token payload, a query is made to the database to ensure that the user that sent the request currently has an active conversation with the chatbot. If not a HTTP 404 error is sent, otherwise an asynchronous request is sent to the classifier service on behalf of the user using httpx’s AsyncClient. This request takes the request body of the original request and adds an extra query parameter of the active ‘conversationid’ needed by the classifier service for adding information to the database. On the endpoint receiving a response from the async request it extracts the information and assigns it to ‘response’ before returning this to the user. The POST ‘/extract’ end point has an identical setup except it sends the request to the Grammar Extractor service instead.

The web-socket endpoint ‘/chat’ has a greater complexity of implementation than the above endpoint definition as it acts as a web-socket proxy between the User Interface and Chatbot services and conducts conversation management. To support this conversation management functionality the additional status, start time and end time fields were added to the conversation Table exemplified in Figure 15 of section 4.6 Database Design. This implementation can be observed in Figure 46.

```
@app.websocket('/chat')
async def websocket(ws_ui: WebSocket, token: str = Query(...), Authorize: AuthJWT = Depends(),
                    db: Session = Depends(get_db)):
    await ws_ui.accept()
    Authorize.jwt_required("websocket",token=token)
    decoded_token = Authorize.get_raw_jwt(token)
    current_user = decoded_token['sub']
    activeConversation=crud.get_active_conversation(db, username=current_user)
    if activeConversation is None:
        try:
            startTime = datetime.now()
            crud.create_conversation(db, username=current_user, startTime=startTime)
            async with websockets.connect(ws_chatbot_url) as ws_bot:
                forward_task = asyncio.create_task(forward(ws_ui, ws_bot))
                reverse_task = asyncio.create_task(reverse(ws_ui, ws_bot))
                await asyncio.gather(forward_task, reverse_task)
        except:
            endTime = datetime.now()
            activeConversation=crud.get_active_conversation(db, username=current_user)
            activeConversation.grammars
            activeConversation.words
            if len(activeConversation.grammars)==0 and len(activeConversation.words)==0:
                crud.delete_conversation(db,conversation_id=activeConversation.conversation_id)
            else:
                difference = endTime - startTime
                minutes = int(divmod(difference.total_seconds(), 60)[0])
                crud.end_conversation(db, username=current_user, endTime=endTime, duration=minutes)
    else:
        error_text="You currently already have an active conversation running. " \
             "Please either continue that conversation or close it before starting a new one"
        await ws_ui.send_text(error_text)
        await ws_ui.close()
```

Figure 46: Chat Web-socket Endpoint Function

This endpoint function has the same ‘Authorize’ and ‘db’ input variables with the same dependencies. It also incorporates: a ‘ws_ui’ variable representing the web-socket connection with the UI service and a ‘token’ variable which matches to a token query parameter that is required in the request URL. This is the only occurrence of the access token being passed in the URL as a query parameter owing to web-socket requests not supporting additional headers. On a request to this endpoint the web-socket handshake process is completed before the access token is extracted from the URL and verified using the ‘Authorize’ variable. If the access token is accepted again using the ‘db’ variable, the ‘get_active_conversation’ function and the username from the access token payload, the database is queried to check if the user has an active conversation. If the user already has an active conversation, an error message is sent through the web-socket before the connection is closed. This is because the current implementation of the FS1 functionality does not support users having multiple conversations with the chatbot at one time. If the user does not have an active conversation, using the ‘create_conversation’ function from crud.py, a new conversation is added to the database and the web-socket proxy process is commenced.

```
async def forward(ws_ui: WebSocket, ws_bot: websockets.WebSocketClientProtocol):
    while True:
        data = await ws_ui.receive_text()
        await ws_bot.send(data)

async def reverse(ws_ui: WebSocket, ws_bot: websockets.WebSocketClientProtocol):
    while True:
        data = await ws_bot.recv()
        await ws_ui.send_text(data)
```

Figure 47: Web-socket Proxy Functions

To implement the web-socket proxy the python in-built asyncio library was leveraged. The first step of the proxy process is the endpoint sending an asynchronous request to establish a separate web-socket connection between the User Authentication service and Chatbot service’s ‘/chat’ endpoint. Once this connection is successfully established two asyncio tasks are created, using the forward and reverse functions highlighted in Figure 47. Referring to Figure 47 both the forward and reverse functions take ‘ws_ui’, the web-socket connection between the User Interface service and the endpoint, and ‘ws_bot’, the web-socket connection between the Chatbot service and the endpoint, as inputs. The forward function starts a loop to receive data from ‘ws_ui’ and sends it to ‘ws_bot’. Reverse does the opposite, also starting a loop but to receive data from ‘ws_bot’ and sends it to ‘ws_ui’. Making these two functions into asyncio tasks and passing them into the asyncio gather method, refer back to Figure 46, allows them to be run in parallel in the background. This facilitates the constant passing of information from one web-socket connection to the other; a web-socket proxy.

This web-socket proxy code is wrapped in a ‘try except’ for the purpose of catching any errors that are thrown during the proxy process and updating the conversation information in the database. The ‘get_active_conversation’ function call in the except clause determines how many word and grammar entries were saved for the current active conversation. If there are none the ‘delete_conversation’ function removes the conversation from the database, ensuring the database is not cluttered with conversations that do not have any words or grammars associated with them. If there are words or

grammars or both, the status of the conversation is updated to Inactive with the end time and duration of the conversation also being added to the database entry through a call to the ‘end_conversation’ function.

5.8 Summary

In this section the implementation of the different development and test environments and the most challenging aspects of the Chatbot, Classifier, Grammar Extractor, User Interface and User Authentication services were documented. Integral sections of the source were highlighted and explained with the codebase structure also being specified. In section 6 the features discussed in this section will be tested.

6 Testing

In this section three different types of testing are performed on the system, unit testing, integration testing and system testing. The unit tests will test the individual endpoints and components of the different micro-services. The integration test will test the communication between the micro-services. Finally, the system test will test the system against the requirements set out in section 3.3 and 3.2 and will also incorporate some validation tests.

6.1 Unit Testing

6.1.1 Unit Testing Environment Docker Compose

As previously mentioned in section 5.2.2 Docker Compose Files the docker-compose.test.yml file contains the testing configurations for the testing environment.

```
services:
  chatbot:
    command: >
      bash -c "pip install requests pytest coverage &&
      python -m coverage run -m pytest &&
      coverage report -m"

  classifier:
    command: >
      bash -c "pip install requests pytest coverage &&
      python -m coverage run -m pytest &&
      rm ./tests/test.db &&
      coverage report -m"

  grammar_extractor:
    command: >
      bash -c "pip install requests pytest coverage &&
      python -m coverage run -m pytest --ignore grammar/tests/test_grammar_rules.py &&
      python grammar/tests/test_grammar_rules.py &&
      rm ./tests/test.db &&
      coverage report -m"

  user_auth:
    command: >
      bash -c "pip install requests pytest asynctest coverage &&
      python -m coverage run -m pytest &&
      rm ./tests/test.db &&
      coverage report -m"
```

Figure 48: Docker Compose Testing Configurations

Unlike for the development environment, there was no need to for volumes to be added, ports to be mapped or environment variables to be set. For each of the python based services a command is run that installs the necessary test dependencies, runs the unit tests, cleans up by deleting the sqlite3 test database that gets generated and displays the coverage report. The test environment is run using the "**docker-compose -f docker-compose.yml -f docker-compose.test.yml --env-file ./.env.test up**" command which starts the services and runs the tests with the services automatically shutting down once testing is completed.

6.1.2 Endpoint Unit Testing

Unit Testing is a type of software testing where individual units of a system are tested in isolation to ensure they are functioning as designed. A unit is the smallest piece of code that can be isolated in a

system, such as a function or a property [65].For each of the micro-service APIs written using FastAPI a set of unit tests were created using Pytest and the built-in FastAPI testing functionality. These tests are examples of the White Box Testing method as the tester knows the details of the source code that is being tested. Mocking is a process used in unit testing to ensure that only the specific unit of code is being tested without relying on the code's external dependencies. Mocking replaces these dependencies with mock objects that simulate the behaviour of the external dependency.

```
def test_classify_endpoint(app: FastAPI, db_session: Session, client: TestClient):
    conversation_id = 1
    with patch('classification_model.word_level_classifier.parser') as mock:
        mock.word_tokenize.return_value =["你", "好"]
        response = client.post("/classify?conversationId=1",
                               json={"message": "你好"})
        assert response.status_code == 200
        assert response.json() == {"classifications": [
            {"word": "你", "level": 1}, {"word": "好", "level": 1}]}

        assert len(db_session.query(Word).filter(Word.word == "你",
                                                 Word.conversation_id == conversation_id).all()) == 1
        assert len(db_session.query(Word).filter(Word.word == "好",
                                                 Word.conversation_id == conversation_id).all()) == 1
```

Figure 49: Classifier Service Endpoint Unit Test

Figure 49 is an example of one of these unit test found in test_main.py in the tests directory of the classifier service. This checks that the '/classify' endpoint returns the correct response on a valid request. For the test function the inputs map to a Pytest feature called fixtures. When Pytest starts to run a test, it looks at the input parameters of the test function then searches for a fixture with the same name. Once found, Pytest runs the fixture, takes the returned value or values and passes them to the test function as the corresponding input parameter [66]. 'app', 'db_session' and 'client' are custom fixtures defined in conftest.py.

```
@pytest.fixture()
def client(app: FastAPI, db_session: Session) -> Generator[TestClient, Any, None]:

    def _get_test_db():
        try:
            yield db_session
        finally:
            pass

    app.dependency_overrides[get_db] = _get_test_db
    with TestClient(app) as client:
        yield client
```

Figure 50: Pytest Client Fixture

Figure 50 shows the definition of the client fixture which actually calls the 'app' and 'db_sesision' fixtures. Referring back to Figures 45 and 46 the 'get_db' function is used as a dependency to retrieve a session connection to the database service. For the unit tests this is not the desired behaviour as, stated previously, unit tests should not have any external dependencies. The client fixture defines the '_get_test_db' function that returns the session connection to the test sqlite3 database provided by the db_session. The fixture then takes the instance of the FastAPI application being tested returned by the

‘app’ fixture, and replaces ‘get_db’ dependency function with the ‘_get_test_db’ function. This means that any database operation performed by the test application will use the test database rather than the actual database in the database service. Finally, the fixture create an instance of the FastAPI TestClient which will be used to make requests to the test application. This ‘TestClient’ gets returned to the test function as the client input parameter.

Figure 49 also display an example of mocking. When making a request to the ‘/classify’ endpoint part of the response process includes sending a request to the coreNLP service. This is an external dependency that should not be accessed during this unit test. Therefore the parser variable, which is normally an instance of a ‘StanfordCoreNLP’ object from stanfordcorenlp library, is replaced with a mock object. Using this mock object the response of the ‘word_tokenize’ function is set to what would be the response from the request without sending one. Using the client input a valid request is sent to the ‘/classify’ endpoint of the test application. Note that the test database is filled with dummy data such as a conversation whose id is used in the request to the endpoint. Next the response from the endpoint is checked to ensure that the expected status code and response body is returned. The test database is also checked to ensure that the correct entries to the database were made.

```
def test_create_grammar(db_session: Session):
    assert len(db_session.query(Grammar).all()) == 0
    conversation_id = 1
    crud.create_grammar(db_session, conversation_id=conversation_id,
                        pattern="没有/没 (+ Obj.)", level='1')

    grammar = db_session.query(Grammar).all()
    assert len(grammar) == 1
    assert grammar[0].pattern == "没有/没 (+ Obj.)"
    assert grammar[0].level == "1"
    assert grammar[0].conversation_id == conversation_id
```

Figure 51: Grammar Extractor Crud Function Unit Test

Figure 51 is the unit test for the ‘create_grammar’ function used in the Grammar Extractor service. All the services that interact with the database service, the Classifier, Grammar Extractor and User Authentication Services, contain a crud.py file which contain a set of functions used by the endpoints. These functions complete the necessary database operations needed by the service i.e. the ‘create_grammar’ function adds a new entry to the Grammar Table in the database. The test function first performs a sanity check that there are no entries in the Grammar Table of the test database. Next the ‘create_grammar’ function is used to add a new entry to the database before a number of checks are performed to confirm this process was complete correctly. The test confirms that only one entry was added to the Grammar Table and all the values entered are as expected.

6.1.3 Chatbot Service Additional Unit Testing

An extra set of unit tests were written to test the sentence similarity functionality demonstrated in section 5.3 Chatbot Service. These tests validate the ‘compare_to_history’, ‘compare_sentences’ and ‘get_random_question’ functions. The ‘compare_to_history’ test ensures the function produces the expected outputs. The ‘compare_sentences’ test verifies that sentence comparison calculation is being performed correctly. The ‘get_random_question’ test mocks the ‘questions’ variable, replacing it with a

list containing a single test question prior to ensuring the function correctly adds the random question to the history list and produces the expected outputs.

```
def test_compare_to_history():
    sent1 = tokenizer.convert_tokens_to_ids(tokenizer.tokenize('我很高兴'))
    results = compare_to_history([sent1],'我 很 高 兴',tokenizer)
    assert results == {'possible_out_text': '我 很 高 兴',
                       'prev_texts': ['我 很 高 兴'], 'similarities': [1.0]}

def test_compare_sentences():
    sent1 = tokenizer.convert_tokens_to_ids(tokenizer.tokenize('我很高兴'))
    prev_text, similarity = compare_sentences('我 很 高 兴', sent1,tokenizer)
    assert similarity == 1.0

def test_get_random_question():
    with patch('chatbot_utils.sentence_simlarity.questions', ['你怎么样？']):
        sent1 = tokenizer.convert_tokens_to_ids(tokenizer.tokenize('我很高兴'))
        history, out_text = get_random_question([sent1],tokenizer)
        assert history ==[[10, 87, 155, 426],[23, 190, 38, 75, 56]]
        assert out_text == '你怎么样？'
```

Figure 52: Chatbot Similarity Functionality Unit Tests

6.1.4 Classifier Service Additional Testing

A set of extra unit tests were written to test the functions of the WordLevelClassifier class which the service uses to interact with the classification model.

```
def test_WordLevelClassifier_encode():
    from classification_model.word_level_classifier import WordLevelClassifier
    classifier = WordLevelClassifier()
    input_ids, attn_mask =classifier.encode('你')
    assert_equal(input_ids.numpy(), [[101, 872, 102,   0,   0,   0]])
    assert_equal(attn_mask.numpy(), [[1, 1, 1, 0, 0, 0]])

def test_WordLevelClassifier_preprocess():
    from classification_model.word_level_classifier import WordLevelClassifier
    classifier = WordLevelClassifier()
    words = classifier.preprocess('你好吗？')
    assert words == ['你','好','吗']

def test_WordLevelClassifier_classify():
    with patch('classification_model.word_level_classifier.parser') as mock:
        mock.word_tokenize.return_value =["你", "好"]
        from classification_model.word_level_classifier import WordLevelClassifier

        classifier = WordLevelClassifier()
        classifications = classifier.classify('你好')
        assert classifications ==[{"word": "你", "level": 1}, {"word": "好", "level": 1}]
```

Figure 53: WordLevelClassifier Class Unit Tests

The first tests validates that the text encoding process is functioning correctly. The second test verifies that the ‘preprocess’ function is correctly removing punctuation from the input sentences. The final test

mocks the parser variable and its ‘word_tokenize’ function response while ensuring that the classification process produces the expected output in the correct format.

When testing the performance of a classification model there are other useful metrics other than just the overall accuracy on the test dataset that can be used to evaluate the model. Precision, recall and f-score are all examples of popular alternatives. The precision score of the model signifies the proportion of positive identifications that were correct, the recall signifies the proportion of ‘true positives’ that were identified correctly. The F-score combines the precision and recall scores with the following formula:

$$Fscore = 2 \times \frac{precision \times recall}{precision + recall}$$

To calculate all of these metrics the sklearn classification report function applied and its output is displayed below.

	precision	recall	f1-score	support
1	0.62	0.33	0.43	15
2	0.32	0.53	0.40	15
3	0.27	0.33	0.30	30
4	0.35	0.53	0.42	60
5	0.49	0.43	0.46	130
6	0.84	0.76	0.80	250
accuracy			0.60	500
macro avg	0.48	0.49	0.47	500
weighted avg	0.63	0.60	0.61	500

Figure 54: Classification Model Classification Report

Figure 54 provides insight into how well the model is performing trying to classify each class. The model performed best classify class 6, with it receiving the highest precision, recall and hence f1-score for this class. Considering the HSK dataset had a large number of class 6 examples it is to be expected that the model would score this high. However, the model performed poorly on all the other classes with it being unable to achieve and f1-score over 0.5 for any of them. This brings to light that even after using the oversampling technique the HSK dataset was too unbalanced to produce a high performance classification model with the fine-tuning approach.

6.1.5 Grammar Extractor Service Additional Unit Testing

The Grammar Extractor service contains an additional set of unit tests to test the accuracy of the regex patterns written for each grammar rule and to test the 'parseSentence' and 'extractGrammar' functions used during the extraction process.

```
def test_accurracy():
    correct=0
    count=0
    for i, grammarRule in enumerate(grammarRules):
        if type(grammarRule['rules']) is str:
            sentences=test_sentences[i]
            p=re.compile(grammarRule['pattern'])
            for sentence in sentences:
                count+=1
                matches=p.findall(sentence)
                if len(matches)!=0:
                    correct+=1
        else:
            for j,pattern in enumerate(grammarRule['rules']):
                sentences=test_sentences[i][j]
                p=re.compile(pattern['pattern'])
                for sentence in sentences:
                    count+=1
                    matches=p.findall(sentence)
                    if len(matches)!=0:
                        correct+=1
        return round((correct/count)*100,2)
```

Figure 55: Grammar Regex Patterns Test Function

Figure 55 illustrates the testing function used to test the grammar regex patterns. For the purpose of testing a set of example sentences were taken from the Allset grammar wiki relating to each grammar pattern to form a test dataset. Each grammar regex pattern was then tested against these test sentences to see if the regex pattern was being activated successfully. Using this technique, the defined grammar regex patterns have an accuracy of 96.15%

```
def test_parseSentence():
    with patch('grammar.grammarUtils.parser') as mock:
        mock.pos_tag.return_value = [
            ('我', 'PN'), ('没有', 'VE'), ('问题', 'NN'), ('。', 'PU')]
        from grammar.grammarUtils import parseSentence
        pos_sent = parseSentence("我没有问题。")
        assert pos_sent == "我PN没有VE问题NN。PU"

def test_extractGrammar_single_pattern():
    with patch('grammar.grammarUtils.parser') as mock:
        mock.pos_tag.return_value = [('太', 'AD'), ('好', 'VA'), ('了', 'SP')]
        from grammar.grammarUtils import extractGrammar
        grammarExtracted, _ = extractGrammar("太好了")
        assert grammarExtracted == [{'rule': '太 + Adj. + 了', 'level': 1}]
```

Figure 56: Grammar Extractor Parse Sentence and Extract Grammar Test Functions

Figure 56 shows the functions used to test the ‘parseSentence’ and ‘extractGrammar’ functions. Both tests also use mocking to again replace the parser variable with a mock object, but this time the ‘pos_tag’ function response is mocked. The ‘test_parseSentence’ test function verifies that the POS sentences are being formed correctly and the ‘test_extractGrammar_single_pattern’ test function verifies that the expected output is returned from the grammar extraction process.

6.1.6 Test coverage

For each of the services containing unit tests, the Chatbot, Classifier, Grammar Extractor and User Authentication services, the coverage of the tests were calculated. The coverage for each of the services can be seen in the subsequent Figures. Every service has over 90% test coverage with the average coverage across the services being 96% highlighting the comprehensiveness of the unit testing

```
Name                                                  Stmts   Miss  Cover   Missing
-----------------------------------------------------------------------------------
__init__.py                                               0      0   100%
chatbot_utils/__init__.py                                 0      0   100%
chatbot_utils/sentence_simlarity.py                      38      3    92%   16-18
chatbot_utils/tests/test_sentence_similarity.py          23      0   100%
tests/__init__.py                                         0      0   100%
tests/test_main.py                                        0      0   100%
-----------------------------------------------------------------------------------
TOTAL                                                    61      3    95%
```

Figure 57: Chatbot Service Unit Test Coverage

```
Name                                                  Stmts   Miss  Cover   Missing
-----------------------------------------------------------------------------------
__init__.py                                               0      0   100%
classification_model/__init__.py                          0      0   100%
classification_model/bert_classifier.py                  15      0   100%
classification_model/tests/__init__.py                    0      0   100%
classification_model/tests/test_classifier.py            21      0   100%
classification_model/word_level_classifier.py            43      1    98%   13
db/__init__.py                                            0      0   100%
db/crud.py                                                8      0   100%
db/database.py                                           16      1    94%   13
db/models.py                                             25      0   100%
dependencies.py                                           7      5    29%   4-10
main.py                                                  30      3    90%   40-43
tests/__init__.py                                         0      0   100%
tests/conftest.py                                        43      0   100%
tests/test_crud.py                                       20      0   100%
tests/test_main.py                                       33      0   100%
-----------------------------------------------------------------------------------
TOTAL                                                   261     10    96%
```

Figure 58: Classifier Service Unit Test Coverage

```
Name                                    Stmts   Miss  Cover   Missing
-----------------------------------------------------------------------
__init__.py                                 0      0   100%
db/__init__.py                              0      0   100%
db/crud.py                                  8      0   100%
db/database.py                             17      1    94%   13
db/models.py                               25      0   100%
dependencies.py                             7      5    29%   4-10
grammar/__init__.py                         0      0   100%
grammar/grammarRules.py                    23      0   100%
grammar/grammarUtils.py                    34      7    79%   8, 24-29, 39
grammar/tests/__init__.py                   0      0   100%
grammar/tests/test_grammar_utils.py        13      0   100%
main.py                                    29      3    90%   37-40
tests/__init__.py                           0      0   100%
tests/conftest.py                          43      0   100%
tests/test_crud.py                         20      0   100%
tests/test_main.py                         32      0   100%
-----------------------------------------------------------------------
TOTAL                                     251     16    94%
```

Figure 59: Grammar Extractor Service Unit Test Coverage

```
Name                  Stmts   Miss  Cover   Missing
-----------------------------------------------------
__init__.py               0      0   100%
api/__init__.py           0      0   100%
api/api_utils.py          7      0   100%
api/schemas.py           23      0   100%
db/__init__.py            0      0   100%
db/crud.py               36      0   100%
db/database.py           16      1    94%   13
db/models.py             33      0   100%
dependencies.py           7      5    29%   4-10
main.py                 104     11    89%   57, 119-125, 174-177
tests/__init__.py         0      0   100%
tests/conftest.py        37      0   100%
tests/test_crud.py      114      0   100%
tests/test_main.py      182      2    99%   166, 233
-----------------------------------------------------
TOTAL                   559     19    97%
```

Figure 60: User Authentication Service Unit Test Coverage

6.2 Integration Testing

Integration testing is a type of software testing where individual units of a system are combined and tested together to ensure that the different units are interacting correctly . To perform the integration testing for the prototype system the pydotenv, pytest and testcontainers python libraries were utilised. To run the integration tests these libraries had to be installed locally with python 3.7. The testcontainers python library is the python port for the Testcontainers Java library which is designed specifically to make Data access layer and Application integration testing easier by using docker containers [67]. Using the docker-compose.yml and docker-compose.override.yml files, and the .env.int.test file to set the environment variables, an integration testing environment is created and used to test that the different services are communicating with each other as expected. These integration tests are defined in the intergration_test.py file which contains five integration tests: one to check that the database service is accessible in the network, another to check that the core_nlp server service is accessible, another to check that the user authentication service is communicating successfully with the database service, another to check that the grammar extractor service is communicating successfully with the database and coreNLP services and finally one to check that the classifier service is communicating successfully with the database and coreNLP services. Figure 61 shows the grammar extractor service integration test.

```
def test_grammar_extractor_db_and_coreNLP_access():
  compose, conn = setup_module()
  cur = conn.cursor()

  response = requests.post("http://localhost:8002/extract?conversationId=1",
                           json={"message": "我没有问题。"})

  assert response.status_code == 200
  assert response.json() == {"grammars": [
    {"rule": "没有/没 (+ Obj.)", "level": 1}, {"rule": "Subj. + Verb (+ Obj.) ", "level": 1}]}

  cur.execute("SELECT * FROM grammar WHERE conversation_id = 1")
  grammars = cur.fetchmany(3)
  assert grammars[1][0] == 2
  assert grammars[1][1] == 1
  assert grammars[1][2] == '没有/没 (+ Obj.)'
  assert grammars[1][3] == '1'

  assert grammars[2][0] == 3
  assert grammars[2][1] == 1
  assert grammars[2][2] == 'Subj. + Verb (+ Obj.) '
  assert grammars[2][3] == '1'
  cur.close()
  teardown_module(compose)
```

Figure 61: Grammar Extractor Integration Test

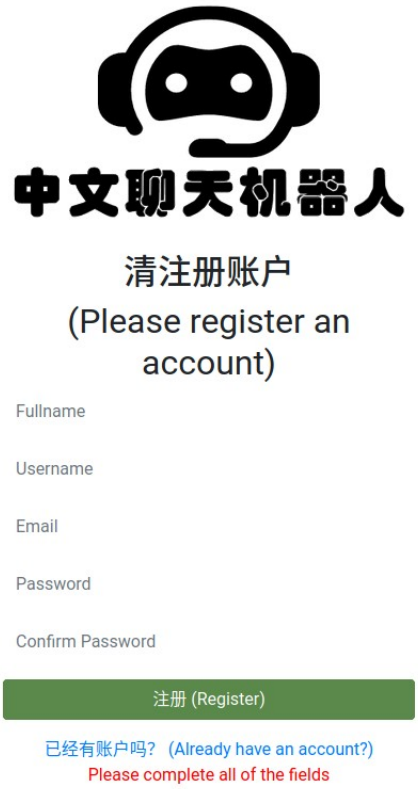
The call to the setup_module function creates a new instance of the application in the integration testing environment and returns the testcontainers' DockerCompose object associated with this instance and a connection to the database in this instance. Next a request is made to the '/extract' endpoint of the grammar_extractor service referencing an active conversation added to the database as dummy data. The response to this request is checked to ensure that it was completed successfully and returns the expected result. This successful response validates that the grammar-extractor service communicated with the coreNLP service as the grammar extraction process could not have been completed successfully otherwise. Next, successful communication between the grammar extractor and database

services is verified by using the database connection to check that the appropriate information was added to the database. Finally, the connection to database is closed and the test instance of the application is stopped.

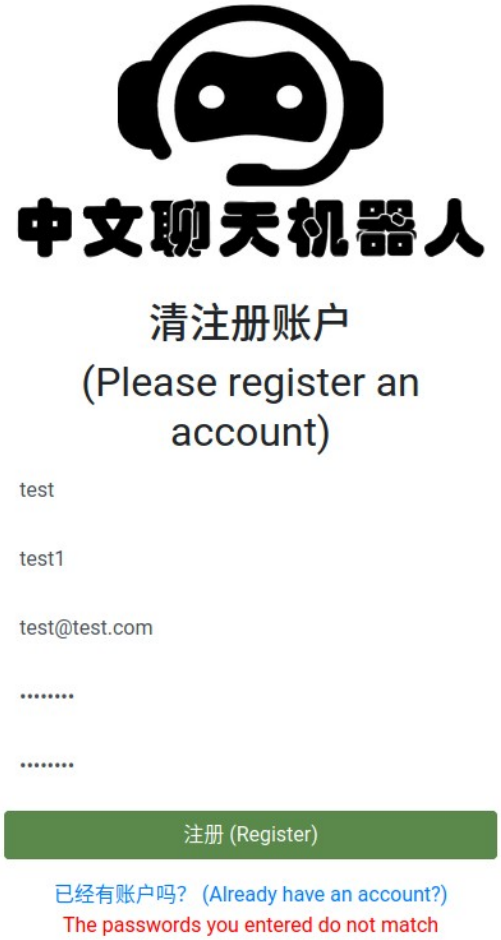
6.3 System & Validation Testing

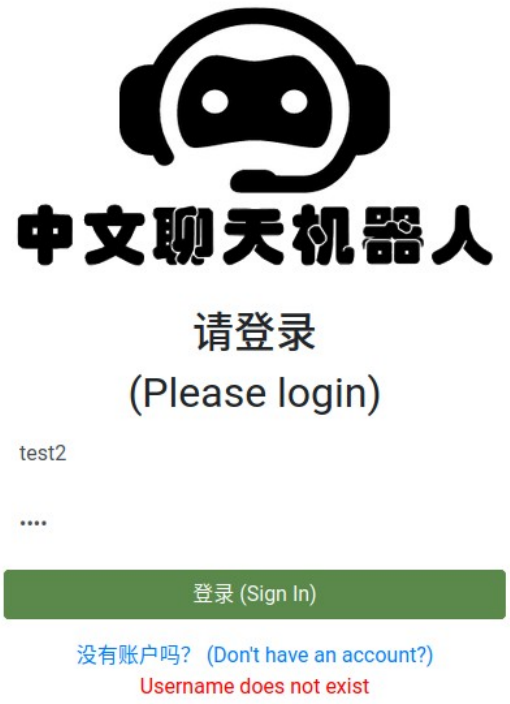
In this section a combination of system testing and validation testing will be performed on the system with the outcomes presented in the Table below

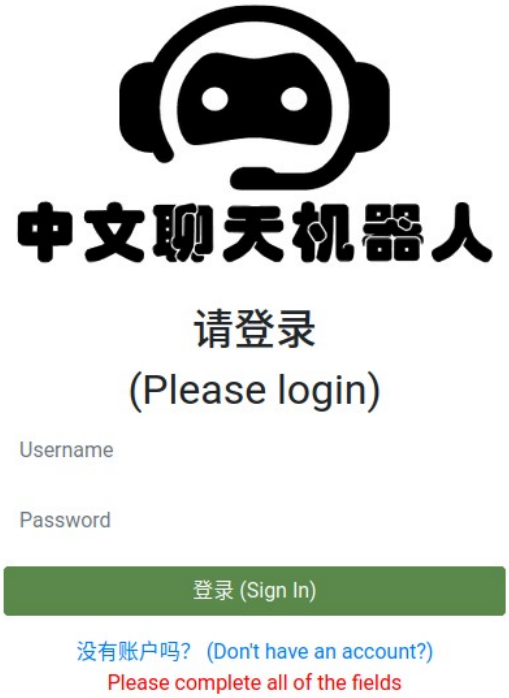
Req ID	Expected Outcome	Actual Outcome	Requirement Met (Y/N)
FM1	The user registers an account and it is added to the database. The user is then logged into the system and redirected to the homepage	A test user was created using the details below The test user account information was added to the database and the browser then redirected the user to the homepage	Y

FM1 V1	Validation: Error message shown when any of the fields are left incomplete	 Correct error message shown	Y
FM1 V2	Validation: Error message shown when a user tries to register with a username already in use.	 Correct error message shown	Y

FM1 V3	Validation: Error message shown when the password entered does not meet the strength requirements of the system	 Correct error message shown	Y
FM1 V4	Validation: Error message shown when invalid email entered	 Correct error message shown	Y

FM1 V5	Validation: Error message shown when passwords entered do not match	 Correct error message shown	Y
FM2	A user is able to log into the system	The user logs into the system using their details and redirected to the homepage 	Y

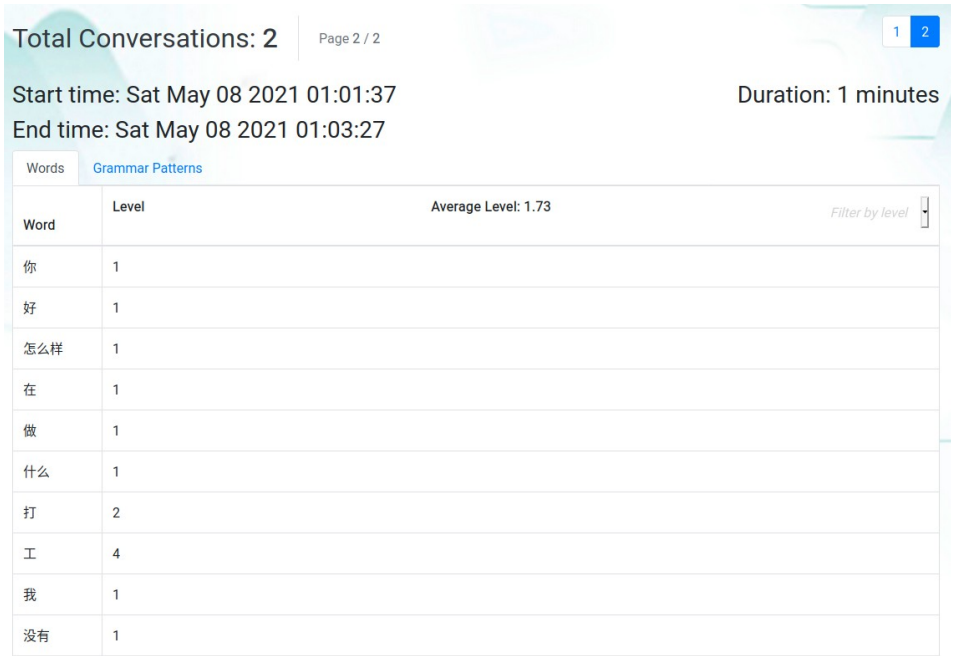
FM2 V1	Validation: Error message shown when username that does not exist in the database is entered	 Correct error message shown	Y
FM2 V2	Validation: Error message shown when incorrect password is entered	 Correct error message shown	Y

FM2 V3	Validation: Error message shown when any of the fields are left incomplete	 Correct error message shown	Y
FM3	The user can log out of the system	After pressing the logout button on the home page or navigation bar the user is logged out of the system	Y
FM4	A user can have a conversation with the chatbot	 The user sends a message to the chatbot and the chatbot responds	Y

FM4 V1	Validation: Error message shown when message is sent in English	 Correct error message shown	Y
FM4 V2	Validation: Error message shown when user tries to start multiple conversation with the chatbot	 Correct error message shown	Y

<table>
<tr><td>FM5</td><td>A user must be able to see information on the words used in their messages</td><td>Word Breakdown
<table><tr><th>Word</th><th>Level</th></tr><tr><td>了</td><td>1</td></tr><tr><td>可惜</td><td>4</td></tr><tr><td>太</td><td>1</td></tr><tr><td>吗</td><td>1</td></tr><tr><td>天气</td><td>1</td></tr><tr><td>深圳</td><td>4</td></tr><tr><td>哪里</td><td>1</td></tr><tr><td>在</td><td>1</td></tr><tr><td>工作</td><td>1</td></tr><tr><td>做</td><td>1</td></tr></table>1 2 ></td><td>Y</td></tr>
<tr><td>FM6</td><td>A user must be able to see information on the grammar patterns used in their messages</td><td>Grammar Breakdown
<table><tr><th>Grammar Pattern</th><th>Level</th></tr><tr><td>太 + Adj. + 了</td><td>1</td></tr><tr><td>Subj. + Verb + 什么 + (Noun)?</td><td>1</td></tr><tr><td>Subj. + Verb (+ Obj.)</td><td>1</td></tr></table></td><td>Y</td></tr>
</table>

<table>
<tr><td>FM7</td><td>A user must be able to see the average proficiency level of their vocabulary</td><td>Word Breakdown
<table><tr><th>Word</th><th>Level</th></tr><tr><td>了</td><td>1</td></tr><tr><td>可惜</td><td>4</td></tr><tr><td>太</td><td>1</td></tr><tr><td>吗</td><td>1</td></tr><tr><td>天气</td><td>1</td></tr><tr><td>深圳</td><td>4</td></tr><tr><td>哪里</td><td>1</td></tr><tr><td>在</td><td>1</td></tr><tr><td>工作</td><td>1</td></tr><tr><td>做</td><td>1</td></tr></table>1 2 >
Average Level: 1.56</td><td>Y</td></tr>
<tr><td>FM8</td><td>A user must be able to see the average proficiency level of their grammar</td><td>Grammar Breakdown
<table><tr><th>Grammar Pattern</th><th>Level</th></tr><tr><td>太 + Adj. + 了</td><td>1</td></tr><tr><td>Subj. + Verb + 什么 + (Noun)?</td><td>1</td></tr><tr><td>Subj. + Verb (+ Obj.)</td><td>1</td></tr></table>Average Level: 1</td><td>Y</td></tr>
<tr><td>FS1</td><td>A user should be able to see word and grammar information from previous conversations</td><td>Total Conversations: 2 Page 2 / 2 1 2
Start time: Sat May 08 2021 01:01:37 Duration: 1 minutes
End time: Sat May 08 2021 01:03:27
Words Grammar Patterns
<table><tr><th>Grammar Pattern</th><th>Level</th></tr><tr><td>Subj. + (正)在 + Verb + Obj.</td><td>2</td></tr><tr><td>Subj. + Verb (+ Obj.)</td><td>1</td></tr><tr><td>Subj. + Verb + 什么 + (Noun)?</td><td>1</td></tr><tr><td>Subj. + 没有/ 没 + Verb</td><td>1</td></tr><tr><td>Option A + 还是 + Option B ?</td><td>1</td></tr></table>Average Level: 1.2 Filter by level</td><td>Y</td></tr>
</table>

			
N1	The user will access the system through a web client	Works as expected. Refer to Appendix C	Y
N2	Response from chatbot within 5 seconds	Works as expected. Demonstrated in the demo video	Y
N3	User input analysis within 25 seconds	Works as expected. Demonstrated in the demo video	Y
N4	All user requests will be authenticated	Works as expected. Demonstrated in the demo video	Y

6.4 Summary

In this section the different tests carried out on the prototype system were examined and their results presented. All the tests implemented were passed by the prototype system, proving that the system was developed to a high standard. These results will be used in section 7 Evaluation and Conclusion to critique the success of the project and propose possible areas for future exploration.

7 Evaluation & Conclusion

This section will present a multifaceted evaluation of the overall success of the project. This project will first be assessed against its objectives prior to the planning of the project and prototype system evaluation. Possible future work will also be discussed before a final conclusion on the success of the project is drawn.

7.1 Evaluation of project against objectives

Section 6 documents the testing process of the prototype system displaying all the functional 'must', some functional 'should' and all non-functional requirements were implemented successfully. The results of these test will be evaluated further in section 7.3. However, when evaluating the success of this project against the objects set out in section 1.3 Aims and Objects is also necessary.

Objective	Description	Achieved (Y/N)
Objective 1	Explore some of the existing implementations of Chinese chatbots used in language learning	Y
Objective 2	Research how Chinese proficiency is assessed and some of the existing Chinese proficiency assessments	Y
Objective 3	Decided on a set of functional and non-functional requirements for the project	Y
Objective 4	Produce designs for the different components of the system based on the defined requirements	Y
Objective 5	Develop a Chinese Chatbot	Y
Objective 6	Develop a machine learning classification model that can determine the proficiency level of a Chinese word	Y
Objective 7	Develop a NLP pipeline that can extract grammar patterns from Chinese text and assign them a proficiency level	Y
Objective 8	Develop a proficiency improvement automated feedback system	N
Objective 9	Develop a web application combining these different components	Y
Objective 10	Conduct a robust set of tests to evaluate the functionality of the system	Y
Objective 11	Evaluate the success of the project and propose possible improvements	Y

Objects 1 entailed exploring some of the existing implementation of chatbots in language learning. This was carried out in section 2.1 where the Mondly and Eggbun Education chatbots were critically reviewed, with their core features highlighted and compared. This object was explored even further in section 2.2 where a comprehensive assessment of the different chatbot types was fulfilled, enabling the determination of the most suitable chatbot type for the project, a generative chatbot.

Objective 2 encompassed researching some to the existing assessments of Chinese proficiency. In section 2.3 the two most popular Chinese proficiency assessments, HSK and TOCFL, were analysed.

The vocabulary and grammar criteria for each level of proficiency in each assessment was documented before the two assessments were compared and contrasted.

Using the results from the completion of object 1 and 2 along with the related research conducted in section 2.4, objective 3 was completed, deriving a set of functional and non-functional for the prototype system. Reviewing related related research papers aided in the realisation of how technically vast the scope of the project was. This prompted the reduction of the project scope in the form of the MoSCoW prioritization ‘Must’ and ‘Should’ requirements ensuring that at the very least an MVP would be produced by the end of the project.

Objective 4 was to produce a set of designs for the different components of the system based on the defined requirements. The successful completion of this objective can be observed in section 4 of this documentation where the designs of the system are discussed in-depth. The architecture of the different machine learning models that were incorporated into the the system, use case and activity diagrams presenting system functionality, database diagrams informing how data will be structured and saved in the system and the system architecture were just some of the topics discussed in this section. These designs provided a clear roadmap to follow when implementing the prototype system and also aided in the implementation process by allowing the implications of design and technology decisions made to be understood beforehand.

Objectives 5, 6 and 7 required the completion of the core components of the prototype system, the chatbot, classification model and grammar extractor. The completion of these objectives is displayed in sections 5.3, 5.4 and 5.5 of the implementation section with the most challenging aspects of each of these objectives being exhibited, and the resulting solutions explained.

Objective 8 required the develop a proficiency improvement automated feedback system but, owing to the time constraints of the project, the objective was not completed. However, this was expected and is the reason why this objective was included as one of the non-essential ‘Should’ requirements of the system.

Objective 9 entailed developing a web application combining these different components created from objectives 5,6 and 7. This objective was completed successfully with the challenging aspects of it’s implementation examined in section 5.6. Evidence of it’s implementation can also be viewed in the screenshots accompanying the system testing in section 6.3 and the demo video submitted along side the project.

Objective 10 was to test the application that was developed. Evidence of the full testing process can be found in Section 6 Testing. The general software testing fundamentals were followed, with the completion of unit, integration and system testing allowing a robust evaluation of the functionality included the prototype system that was created. To improve the testing even further an additional set of tests could have been carried out. For example load testing could have been carried out to ensure that the current system architecture does not create a bottle neck at the User Authentication service or user acceptance testing to get user feedback about the system. These tests were not included currently as a result of the time constraints and also considering that this is the first iteration of the system. However, for future iterations these tests would be essential to encourage the improvement of the system as a whole.

The final objective was to evaluate the success of this project which is being completed in the current and subsequent sections.

7.2 Evaluation of planning

The planning of the project, although initially thought to be thorough, still has room for improvement. The Gantt chart outlined in Appendix A is evidence of the careful consideration put into creating a plan, with time also being allocated for the January exam season plus other module deadlines. However, although a functioning prototype was produced the quality of the classification model and incomplete grammar extractor leaves much to be desired. Even after reducing the project scope the workload of the project was still very high with several milestones throughout the course of the project over running. This can be attributed to a underestimation of how long particular development tasks would take i.e. a large amount of time was spent trying to improve the performance of the classification model owing to a lack of software development experience. In the future more time will be allocated for the learning of new programming languages and technologies to ensure the same mistakes are not repeated.

Another planning improvement that will be considered is to ensure that time is allocated for other responsibilities. Although the Gantt chart considered other module's exam dates and deadlines it did not take into account keeping up with the weekly content for the modules. In the future a set number of days will be allocated for each module and the project to ensure that none of the other modules are neglected.

7.3 Evaluation of the system

As a whole the prototype system works as expected fulfilling all the 'Must' requirements and the additional 'Should' requirement FS1. The current implementation should also be fairly simple to deploy on most cloud platforms due to the use of docker. However, before this is performed the implementation of at least the 'Should' requirements centring around user accounts, FS2, FS3 and FS4 should be completed to provide a better user account experience.

The performance of the classification model was unsatisfactory producing a test accuracy of just 60%. Looking at the classification report it is clear that even after using the oversampling method the classes were simply too unbalanced to train a high performing classification model. However, since the creation of the current classification model, China's Ministry of Education announced the official standards for the new HSK exam. The classes in the vocabulary list for this new standard are a lot more balanced: Level 1 now has 500 words, level 2 has 722, level 3 has 973, level 4 has 1000, level 5 has 1071 and level 6 has 1140. Training the classification model with this new vocabulary list should therefore be able to produce a classification model with better performance. It is also possible that the data set is too small to be use with a pre-trained model like Bert to produce a satisfactory classification model for this task, an issue the Pasa et al paper [68] proposes a possible solution for. In this paper a Convolutional Neural Network (CNN) is created that is able to produce competitive performance on the task of X-ray Tuberculosis image classification. This paper is of interest as these results were achieved training on a dataset even smaller than the HSK data set used to train the classification model of the prototype system. The work of this paper could be explored to determine if there are was a way to produce a CNN that has similar degree of performance for this classification task.

The grammar extractor produced a high accuracy during testing of ..%. However, the number of tests cases provided per grammar pattern were not uniform and also not very large and all of the grammar patterns were not implemented so this accuracy may decrease after the full set is completed. In the future a more extensive test set should be created to get a better understanding of how well the current regex patterns are performing. Another issue with the current POS structure is there are instances where the coreNLP POS tagging process produces unexpected tags. For example the word 没有 is at times

tagged as VE or as AD. This is due to the coreNLP POS tagger having a overall accuracy of 93.65% and the unknown word accuracy is 84.84% [69] meaning that there will be instances that incorrect tagging will occur. However, because the current regex patterns depend on the sentence being tagged correctly more time needs to be spent exploring this to find a method to alleviate this issue.

The chatbot model performs it's purpose of conversing with users in Chinese however it does not perform this perfectly. The first issue is the model's lack of persona, which was pre-emptively highlighted in the literature review section. This results in the model producing unsatisfactory answers to personal questions posed to it or occasionally answering the same question differently i.e. says it's a male in one part of the conversation then a female in another part . The second issue is the chatbot's lack of question asking. When conversing with the chatbot the opposite issue occurs than with the Mondly chatbot, the user has full control over the flow of the conversation. For the chatbot to properly perform the projects aim of addressing the language learning plateau for user with an intermediate level of proficiency it needs to be able to ask more probing questions and lead users into discussing more complicated topics. All these issues could be address with the new GPT-3 model produced by openAI. This new model has 175 billion parameters 1000 times larger than the number of parameters used by the GPT model in the current prototype. Appendix D displays a demo of the GPT-3 in the 'playground' of openAI's closed beta and clearly demonstrates the difference in performance between the two models. The GPT-3 model only needs a two inputs: a sentence describing the task the model needs to perform, in the demo the model is told it is a friendly smart AI chatbot that asks lots of questions, and an example user and and model response, the first 'Human' and 'AI' responses in the demo. With just this information it was able to produce a vastly better conversation than the GPT chatbot used in this implementation.

Finally, the automated feedback system was not able to be implemented due to the time constraints of the project. To be able to fully aid user in overcoming the language learning plateau this is a key feature that needs to be added to the system.

7.4 Future Work

The current prototype system provides a basis for the core functionality of the ideal application, but is far from finished. Removing resources, personnel and time constraints, as well as exploring the possible improvements raised in section 7.2, a multitude of additional features could be added.

The first feature that could be introduced is different chatbot difficulty levels. This feature would enable users to control the level of difficulty of the conversation with the chatbot. Lower level chatbots would use simpler vocabulary and grammar patterns and discuss more basic topics. The higher level chatbots would use more advanced vocabulary and grammar patterns and discuss more complex topics. This feature would enable to system to accommodate all users no matter their level of Chinese proficiency. Another feature that could accommodate lower proficiency users is functionality that could convert messages to Pinyin like the Mondly chatbot reviewed in section 2.1.

Another feature that could be implement is a flash card feature. During a user's conversation with the chatbot additional information could be collected such as words that user did not know the meaning of and words or grammar patterns the user used incorrectly. This information could then be converted into a set flash card that the user could study from to fill the gaps in their learning. Flash cards are a common tool used for foreign language learning and learning in general so support could be added for integration with some popular flash cards such as Anki, Brainscape or Memrise.

The finally additional piece of functionality that could be introduce would be the ability to converse with the chatbot verbally. Currently the system is only able to assess a user's reading and writing due to the current conversation medium but introducing verbal communication will also allow speaking and listening to be assessed. The system could then be able to provide feedback on not just the vocabulary and grammar patterns used but also the user's pronunciation and usage of tones.

7.5 Conclusion

The main aim of this project is to produce a prototype system, that through analysing the conversation between a user and a chatbot in the form of messages, can determine a user's proficiency in Mandarin Chinese. Based on this analysis the system will also make suggestions on areas that the users should concentrate on to improve their proficiency. Due to the complexity of this aim the project scope had to be reduced to accommodate the personnel, resources and time constraints. In terms of the reduced project scope the prototype system cannot be called a complete success mainly due to the poor accuracy produced by the classification model and the incomplete grammar extractor functionality. However, the prototype system provides a solid foundation for the future work of this project that can built upon to produce the desired system functionality and possibly even beyond that.

8 Statement of Ethics

This project was completed closely following the legal, ethical, social and professional standards set by the British Computing Society Code of Conduct (BCOC). The legal obligation outlined by the Computer Misuse (1990), Data Protection (2018) and Copyright, Designs and Patents (1988) Acts were also considered and adhered to throughout the project. This section discusses the ethical issues encountered during the project and justifications for the decisions made regarding them. A SAGE self-check form was also completed and can be found in Appendix B, verifying that no further ethical clearance was necessary for the project.

8.1 Legal Acts

The Computer Misuse Act was introduced to protect personal user data held by organisations, from unauthorized access and modification. This legislation makes it illegal to access computer material without authorisation and or with the intent to: commit a further crime, modify data without authorisation, and to make, supply or obtain anything that can be used in a computer misuse offence. Some of these computer misuse offences include hacking, viruses and computer fraud [70]. To comply with this legislation different security measure were included in the system. Examples of these measures are that all user accounts are password protected, with password strength requirements having to be met before a new account is made. This protects against user accounts being hacked. Furthermore, multiple security measures were implemented to protect the system against XXS and CSFR attacks, covered in greater detail in section 4.5 Security Design

The Data Protection Act, introduced in 1998, protects the confidentiality of personal user data held by organisations. The act dictates a set of ‘data protection principles’ to ensure that user information is:

- used fairly, lawfully and transparently
- used for specified, explicit purposes
- used in a way that is adequate, relevant and limited to only what is necessary
- accurate and, where necessary, kept up to date
- kept for no longer than is necessary
- handled in a way that ensures appropriate security, including protection against unlawful or unauthorised processing, access, loss, destruction or damage

The act was updated in 2018 due to the UK government’s introduction of General Data Protection Regulations (GDPR) which mandated further controls on personal data usage by organisations [71]. The prototype application already adheres to the Data Protection Act, only collecting relevant user data i.e. name and email and not excessive data i.e. collects words and grammar from user conversation with the chatbot but not the actual conversation itself. The prototype also currently holds no private or sensitive information of any individuals but future iterations will have to store private user information such as their name and email. Appropriate measures will have to be put in place to ensure the security of this information. Some measures have already been implemented in the current prototype such as: hashing of user passwords before storage in the database, connections to the database being password protected and the micro-services only accepting requests from specific micro-services in the docker network. However, further security features would have to be implemented once the application is deployed and made accessible to the public. The most important features would be encryption of the database information when at rest and using HTTPS to encrypt requests. Both measures would ensure the confidentiality of user data.

The Copyright, Designs and Patents Act protects the intellectual property of individuals giving them the ability to control how their work is distributed and used [72]. The prototype system does not violate this act as all third-party code used in the application is MIT licensed. This license is commonly used for open-source software and places no restrictions on commercial use.

8.2 Social Considerations

The prototype application and future iterations of the system all will have a positive impact in society as they embody the first principle of the BCO; make IT for everyone. The goal of the system is to aid users in improving their proficiency in Mandarin Chinese which will result in the multiple positive outcomes. Firstly, the application will boost the user's confidence in their language skills. This will eventually lead to them communicating with others in said language hence, improving their communication skills. Secondly, studying a second language has multiple benefits for the individual such as improved memory and multitasking ability. Thirdly, studying a foreign language exposes learners to the culture associated with the language, contributing to greater culture tolerance and an overall more harmonious society. Finally, the ability to speak Chinese is a valuable skill in the current economic climate. Therefore, this applications aids in creating a more skilled work force.

Aside from the positive impacts there are some ways in which the application could be misused. One the main features of the implemented system is proficiency assessment. To try and make this process as accurate as possible, information from official proficiency assessments were used throughout the system. However, this still does not mitigate the fact that this is not an official proficiency assessment so cannot be used to for example apply to job that requires a certain level of proficiency. Rather the user should use the system as form or preparation for completing an official assessment and getting an official accreditation. There is also the potential for the chatbot model to generate offensive language, especially if prompted by an offensive user response. To try and mitigate this issue in future iterations user and chatbot messages could be filtered to ensure they do not contain Chinese profanity, with users who repeatedly send this type of message receiving some sort of ban. This will have to be reviewed thoroughly and tested before the deployment of the application.

9 Bibliography

[1] ‘Mandarin is now rapidly becoming a global language - Opinion - Chinadaily.com.cn’. https://www.chinadaily.com.cn/opinion/2017-10/13/content_33190150.html (accessed Mar. 13, 2021).

[2] ‘Boost to Mandarin teaching in schools’, *GOV.UK*. https://www.gov.uk/government/news/chancellor-announces-boost-to-mandarin-teaching-in-schools (accessed Mar. 13, 2021).

[3] W. M. Morrison, ‘China’s Economic Rise: History, Trends, Challenges, and Implications for the United States’, p. 42.

[4] Y. N. Lee, ‘Here are 4 charts that show China’s rise as a global economic superpower’, *CNBC*, Sep. 24, 2019. https://www.cnbc.com/2019/09/24/how-much-chinas-economy-has-grown-over-the-last-70-years.html (accessed Mar. 13, 2021).

[5] ‘Foreign Language Training’, *United States Department of State*. https://www.state.gov/foreign-language-training/ (accessed Mar. 13, 2021).

[6] Mergermarket, ‘Language Learning Apps Are Seeing A Surge In Interest During The COVID-19 Pandemic’, *Forbes*. https://www.forbes.com/sites/mergermarket/2020/04/15/language-learning-apps-are-seeing-a-surge-in-interest-during-the-covid-19-pandemic/ (accessed Mar. 13, 2021).

[7] J. C. Richards, *Moving beyond the plateau*. Cambridge: Cambridge University Press.

[8] E. K. Horwitz, M. B. Horwitz, and J. Cope, ‘Foreign Language Classroom Anxiety’, *The Modern Language Journal*, vol. 70, no. 2, pp. 125–132, 1986.

[9] L. Fryer and R. Carpenter, ‘Bots as Language Learning Tools’, *Language Learning & Technology*, vol. 10, no. 3, pp. 8–14, Sep. 2006.

[10] ‘Mondly - About Us’. https://www.mondly.com/about (accessed Mar. 15, 2021).

[11] E. Adamopoulou and L. Moussiades, ‘An Overview of Chatbot Technology’, in *Artificial Intelligence Applications and Innovations*, vol. 584, I. Maglogiannis, L. Iliadis, and E. Pimenidis, Eds. Cham: Springer International Publishing, 2020, pp. 373–383. doi: 10.1007/978-3-030-49186-4_31.

[12] K. Nimavat and Prof. T. Champaneria, ‘Chatbots: An Overview Types, Architecture, Tools and Future Possibilities’, *International Journal for Scientific Research & Development*, vol. 5, no. 07, p. 6, 2017.

[13] P. Kucherbaev, A. Bozzon, and G. Houben, ‘Human-Aided Bots’, *IEEE Internet Comput.*, vol. 22, no. 6, pp. 36–43, Nov. 2018, doi: 10.1109/MIC.2018.252095348.

[14] ‘考试介绍--汉语考试服务网’. http://www.chinesetest.cn/gosign.do?id=1&lid=0 (accessed Mar. 13, 2021).

[15] ‘Chinese Proficiency Test: TOP’. https://www.tw.org/tocfl/ (accessed Mar. 13, 2021).

[16] T. Wolf, ‘ How to build a State-of-the-Art Conversational AI with Transfer-Learning’, *Medium*, Sep. 02, 2020. https://medium.com/huggingface/how-to-build-a-state-of-the-art-conversational-ai-with-transfer-learning-2d818ac26313 (accessed Mar. 13, 2021).

[17] E. Dinan *et al.*, ‘The Second Conversational Intelligence Challenge (ConvAI2)’, *ArXiv190200098 Cs*, Jan. 2019, Accessed: Mar. 14, 2021. [Online]. Available: http://arxiv.org/abs/1902.00098

[18] T. Wolf, V. Sanh, J. Chaumond, and C. Delangue, ‘TransferTransfo: A Transfer Learning Approach for Neural Network Based Conversational Agents’, *ArXiv190108149 Cs*, Feb. 2019, Accessed: Mar. 13, 2021. [Online]. Available: http://arxiv.org/abs/1901.08149

[19] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, ‘The Curious Case of Neural Text Degeneration’, *ArXiv190409751 Cs*, Feb. 2020, Accessed: Mar. 14, 2021. [Online]. Available: http://arxiv.org/abs/1904.09751

[20] A. Fan, M. Lewis, and Y. Dauphin, ‘Hierarchical Neural Story Generation’, *ArXiv180504833 Cs*, May 2018, Accessed: Mar. 14, 2021. [Online]. Available: http://arxiv.org/abs/1805.04833

[21] ‘NLPCC 2020 Home’. http://tcci.ccf.org.cn/conference/2020/index.php (accessed Mar. 14, 2021).
[22] ‘NLPCC 2020 Call for Papers’. http://tcci.ccf.org.cn/conference/2020/cfp.php (accessed Mar. 14, 2021).
[23] Y. Wang *et al.*, ‘A Large-Scale Chinese Short-Text Conversation Dataset’, *ArXiv200803946 Cs*, Aug. 2020, Accessed: Mar. 13, 2021. [Online]. Available: http://arxiv.org/abs/2008.03946
[24] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, ‘BLEU: a method for automatic evaluation of machine translation’, in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL ’02*, Philadelphia, Pennsylvania, 2001, p. 311. doi: 10.3115/1073083.1073135.
[25] J. Weston, E. Dinan, and A. H. Miller, ‘Retrieve and Refine: Improved Sequence Generation Models For Dialogue’, *ArXiv180804776 Cs*, Sep. 2018, Accessed: Mar. 14, 2021. [Online]. Available: http://arxiv.org/abs/1808.04776
[26] B. Laufer, ‘The Development of Passive and Active Vocabulary in a Second Language: Same or Different?’, *Appl. Linguist.*, vol. 19, no. 2, pp. 255–271, Jun. 1998, doi: 10.1093/applin/19.2.255.
[27] ‘Chapter 10: MoSCoW Prioritisation’. https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation (accessed Mar. 15, 2021).
[28] ‘What Is SDLC? Understand the Software Development Life Cycle’, *Stackify*, Apr. 08, 2020. https://stackify.com/what-is-sdlc/ (accessed Mar. 13, 2021).
[29] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, ‘Improving Language Understanding by Generative Pre-Training’, p. 12.
[30] A. Vaswani *et al.*, ‘Attention Is All You Need’, *ArXiv170603762 Cs*, Dec. 2017, Accessed: Apr. 23, 2021. [Online]. Available: http://arxiv.org/abs/1706.03762
[31] S. J. Pan and Q. Yang, ‘A Survey on Transfer Learning’, *IEEE Trans. Knowl. Data Eng.*, vol. 22, no. 10, pp. 1345–1359, Oct. 2010, doi: 10.1109/TKDE.2009.191.
[32] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, ‘GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding’, in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, Brussels, Belgium, 2018, pp. 353–355. doi: 10.18653/v1/W18-5446.
[33] L. Xu *et al.*, ‘CLUE: A Chinese Language Understanding Evaluation Benchmark’, in *Proceedings of the 28th International Conference on Computational Linguistics*, Barcelona, Spain (Online), 2020, pp. 4762–4772. doi: 10.18653/v1/2020.coling-main.419.
[34] M. L. Swoyer Steve, ‘Microservices Adoption in 2020’, *O’Reilly Media*, Jul. 15, 2020. https://www.oreilly.com/radar/microservices-adoption-in-2020/ (accessed Apr. 24, 2021).
[35] ‘Empowering App Development for Developers | Docker’. https://www.docker.com/ (accessed Apr. 12, 2021).
[36] ‘What is a Container? | App Containerization | Docker’. https://www.docker.com/resources/what-container (accessed Apr. 12, 2021).
[37] Novick, Vladimir and Gopal,Tanmai, ‘The Ultimate Guide to handling JWTs on frontend clients (GraphQL)’. https://hasura.io/blog/best-practices-of-using-jwt-with-graphql/#jwt_persist (accessed Apr. 08, 2021).
[38] M. Jones, J. Bradley, and N. Sakimura, ‘JSON Web Token (JWT)’, RFC Editor, RFC7519, May 2015. doi: 10.17487/RFC7519.
[39] ‘NIST Special Publication 800-63-3’. /sp800-63-3.html (accessed Apr. 24, 2021).
[40] ‘Virtual DOM and Internals – React’. https://reactjs.org/docs/faq-internals.html (accessed May 13, 2021).
[41] ‘Introducing JSX – React’. https://reactjs.org/docs/introducing-jsx.html (accessed May 13, 2021).
[42] ‘Features - FastAPI’. https://fastapi.tiangolo.com/features/#validation (accessed Apr. 27, 2021).
[43] ‘Hugging Face – The AI community building the future.’ https://huggingface.co/ (accessed May 18, 2021).
[44] ‘Overview’, *CoreNLP*. https://stanfordnlp.github.io/CoreNLP/ (accessed Apr. 27, 2021).

[45] M. O. contributors Jacob Thornton, and Bootstrap, ‘Bootstrap’. https://getbootstrap.com/ (accessed May 15, 2021).
[46] ‘Design’, *Material Design*. https://material.io/design (accessed May 15, 2021).
[47] ‘The Official YAML Web Site’. https://yaml.org/ (accessed May 13, 2021).
[48] G. Hall, *vishnubob/wait-for-it*. 2021. Accessed: May 18, 2021. [Online]. Available: https://github.com/vishnubob/wait-for-it
[49] *thu-coai/CDial-GPT*. thu-coai, 2021. Accessed: May 13, 2021. [Online]. Available: https://github.com/thu-coai/CDial-GPT
[50] L. Shang, Z. Lu, and H. Li, ‘Neural Responding Machine for Short-Text Conversation’, in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Beijing, China, 2015, pp. 1577–1586. doi: 10.3115/v1/P15-1152.
[51] ‘HTCondor - Home’. https://research.cs.wisc.edu/htcondor/ (accessed May 15, 2021).
[52] ‘Computer science facilities | University of Surrey’. https://www.surrey.ac.uk/department-computer-science/study/facilities (accessed May 13, 2021).
[53] ‘Word vectors for 157 languages · fastText’. https://fasttext.cc/index.html (accessed May 13, 2021).
[54] ‘ChatterBot Corpus Documentation — chatterbot-corpus 1.0.0 documentation’. https://chatterbot-corpus.readthedocs.io/en/latest/ (accessed May 13, 2021).
[55] ‘About ChatterBot — ChatterBot 1.0.2 documentation’. https://chatterbot.readthedocs.io/en/sTable/index.html (accessed May 13, 2021).
[56] Y. Liu *et al.*, ‘RoBERTa: A Robustly Optimized BERT Pretraining Approach’, *ArXiv190711692 Cs*, Jul. 2019, Accessed: May 11, 2021. [Online]. Available: http://arxiv.org/abs/1907.11692
[57] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, ‘ALBERT: A Lite BERT for Self-supervised Learning of Language Representations’, *ArXiv190911942 Cs*, Feb. 2020, Accessed: May 11, 2021. [Online]. Available: http://arxiv.org/abs/1909.11942
[58] L. N. Smith and N. Topin, ‘Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates’, *ArXiv170807120 Cs Stat*, May 2018, Accessed: May 11, 2021. [Online]. Available: http://arxiv.org/abs/1708.07120
[59] ‘AllSet Learning’, *AllSet Learning*. https://www.allsetlearning.com (accessed Apr. 13, 2021).
[60] F. Xia, ‘The Part-Of-Speech Tagging Guidelines for the Penn Chinese Treebank 3.0’, p. 44.
[61] A. Janvier, ‘Handling JWT in Admin Apps the Right Way’. https://marmelab.com//blog/2020/07/02/manage-your-jwt-react-admin-authentication-in-memory.html (accessed Apr. 25, 2021).
[62] ‘Promise - JavaScript | MDN’. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise (accessed May 14, 2021).
[63] ‘Context – React’. https://reactjs.org/docs/context.html (accessed Apr. 26, 2021).
[64] ‘State and Lifecycle – React’. https://reactjs.org/docs/state-and-lifecycle.html (accessed May 14, 2021).
[65] ‘Unit Testing’, *SOFTWARE TESTING Fundamentals*, Feb. 04, 2011. https://softwaretestingfundamentals.com/unit-testing/ (accessed May 05, 2021).
[66] ‘How to use fixtures — pytest documentation’. https://docs.pytest.org/en/latest/how-to/fixtures.html (accessed May 05, 2021).
[67] ‘Testcontainers’. https://www.testcontainers.org/ (accessed May 12, 2021).
[68] F. Pasa, V. Golkov, F. Pfeiffer, D. Cremers, and D. Pfeiffer, ‘Efficient Deep Network Architectures for Fast Chest X-Ray Tuberculosis Screening and Visualization’, *Sci. Rep.*, vol. 9, no. 1, p. 6268, Dec. 2019, doi: 10.1038/s41598-019-42557-4.
[69] H. Tseng, D. Jurafsky, and C. Manning, ‘Morphological features help POS tagging of unknown words across language varieties’, p. 8.

[70] E. Participation, ‘Computer Misuse Act 1990’. https://www.legislation.gov.uk/ukpga/1990/18/contents (accessed May 12, 2021).
[71] ‘Data Protection Act 2018’. https://www.legislation.gov.uk/ukpga/2018/12/contents/enacted (accessed May 12, 2021).
[72] E. Participation, ‘Copyright, Designs and Patents Act 1988’. https://www.legislation.gov.uk/ukpga/1988/48/contents (accessed May 12, 2021).

10 Appendices

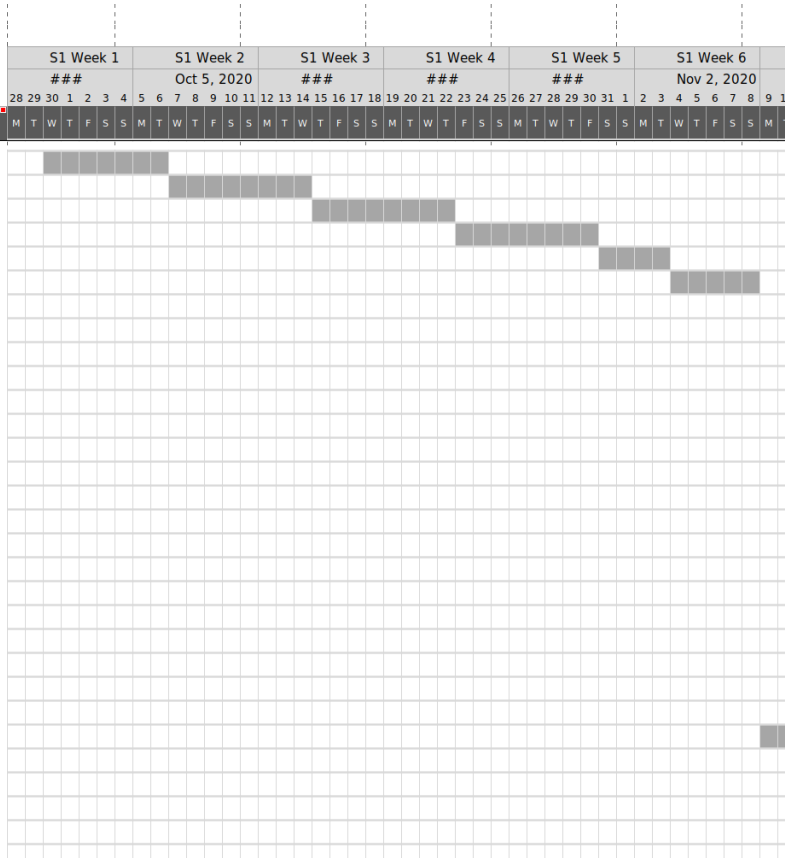
10.1 Append A – Gantt Chart for project

Natural Language Processing and Machine Learning to address the Language Learning Pl

Name: Khalil Russell

Project Start: Mon, 28/09/2020
Today: Tue, 18/05/2021
Week Start: 1

	TASK	PROGRESS	START	END
Prelim Stuff	Preliminary Investigation	100%	30/09/20	06/10/20
	Decide on aims and objectives	100%	07/10/20	14/10/20
	Research potential technologies	100%	15/10/20	22/10/20
	Write up project overview	100%	23/10/20	30/10/20
Lit Review	Do further research on similar Solutions	100%	31/10/20	03/11/20
	Research proficiency assessments	100%	04/11/20	08/11/20
	Review related research papers	100%	16/11/20	23/11/20
Main project	Decide on functional and non-functional requirements	100%	24/11/20	26/11/20
	Design documentation (System architecture + Database)	100%	27/11/20	29/11/20
	Database design	100%	30/11/20	02/12/20
	Develop chatbot micro-service	100%	03/12/20	18/12/20
	Design Classification Model architecture (add to report) and begin develop	100%	19/12/20	03/01/21
	Design NLP pipeline architecture and begin development	100%	25/01/21	11/02/21
	Develop user authentication service	100%	12/02/21	18/02/21
	Develop user interface	100%	19/02/21	25/02/21
	Combine micro-services	100%	26/02/21	04/03/21
	Document implementation	100%	05/03/21	09/03/21
	Draft writing	100%	10/03/21	14/03/21
	Conclusion and Evaluation writing	100%	13/04/21	17/04/21
	Final Touches	100%	18/04/21	18/05/21
Testing	Unit Testing	100%	15/03/21	28/03/21
	Integration Testing	100%	29/03/21	04/04/21
	Bug Fixing	100%	05/04/21	08/04/21
	Document Testing	100%	09/04/21	12/04/21
Other Modules	Working on International Business and Computer Security coursework	100%	09/11/20	15/11/20
	January Exams	100%	04/01/21	24/01/21
	NLP Coursework	100%	26/04/21	02/05/21
	AI Coursework	100%	03/05/21	09/05/21
	Dissertation	100%	10/05/21	16/05/21

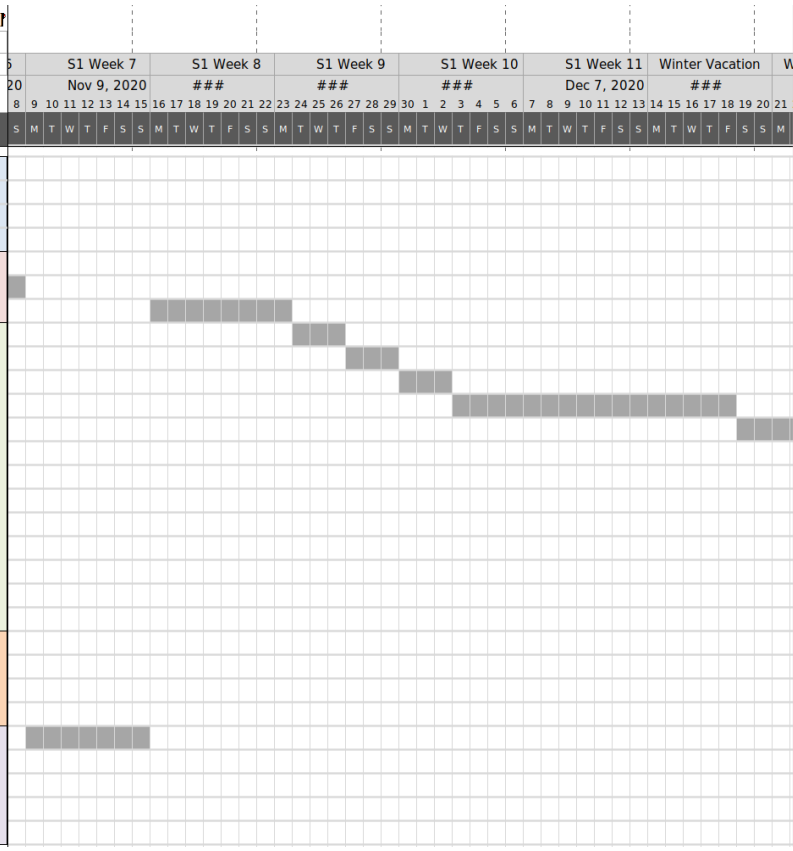


Natural Language Processing and Machine Learning to address the Language Learning Pl

Name: Khalil Russell

Project Start: Mon, 28/09/2020
Today: Tue, 18/05/2021
Week Start: 1

	TASK	PROGRESS	START	END
Prelim Stuff	Preliminary Investigation	100%	30/09/20	06/10/20
	Decide on aims and objectives	100%	07/10/20	14/10/20
	Research potential technologies	100%	15/10/20	22/10/20
	Write up project overview	100%	23/10/20	30/10/20
Lit Review	Do further research on similar Solutions	100%	31/10/20	03/11/20
	Research proficiency assessments	100%	04/11/20	08/11/20
	Review related research papers	100%	16/11/20	23/11/20
Main project	Decide on functional and non-functional requirements	100%	24/11/20	26/11/20
	Design documentation (System architecture + Database)	100%	27/11/20	29/11/20
	Database design	100%	30/11/20	02/12/20
	Develop chatbot micro-service	100%	03/12/20	18/12/20
	Design Classification Model architecture (add to report) and begin develop	100%	19/12/20	03/01/21
	Design NLP pipeline architecture and begin development	100%	25/01/21	11/02/21
	Develop user authentication service	100%	12/02/21	18/02/21
	Develop user interface	100%	19/02/21	25/02/21
	Combine micro-services	100%	26/02/21	04/03/21
	Document implementation	100%	05/03/21	09/03/21
	Draft writing	100%	10/03/21	14/03/21
	Conclusion and Evaluation writing	100%	13/04/21	17/04/21
	Final Touches	100%	18/04/21	18/05/21
Testing	Unit Testing	100%	15/03/21	28/03/21
	Integration Testing	100%	29/03/21	04/04/21
	Bug Fixing	100%	05/04/21	08/04/21
	Document Testing	100%	09/04/21	12/04/21
Other Modules	Working on International Business and Computer Security coursework	100%	09/11/20	15/11/20
	January Exams	100%	04/01/21	24/01/21
	NLP Coursework	100%	26/04/21	02/05/21
	AI Coursework	100%	03/05/21	09/05/21
	Dissertation	100%	10/05/21	16/05/21

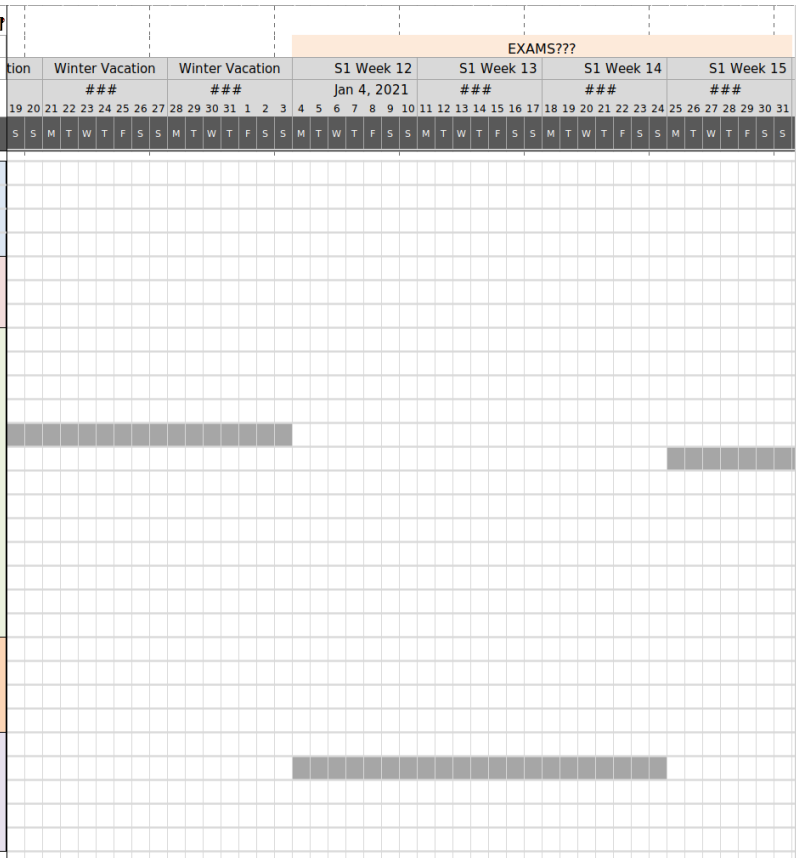


Natural Language Processing and Machine Learning to address the Language Learning Pl

Name: Khalil Russell

Project Start: Mon, 28/09/2020
Today: Tue, 18/05/2021
Week Start: 1

	TASK	PROGRESS	START	END
Prelim Stuff	Preliminary Investigation	100%	30/09/20	06/10/20
	Decide on aims and objectives	100%	07/10/20	14/10/20
	Research potential technologies	100%	15/10/20	22/10/20
	Write up project overview	100%	23/10/20	30/10/20
Lit Review	Do further research on similar Solutions	100%	31/10/20	03/11/20
	Research proficiency assessments	100%	04/11/20	08/11/20
	Review related research papers	100%	16/11/20	23/11/20
Main project	Decide on functional and non-functional requirements	100%	24/11/20	26/11/20
	Design documentation (System architecture + Database)	100%	27/11/20	29/11/20
	Database design	100%	30/11/20	02/12/20
	Develop chatbot micro-service	100%	03/12/20	18/12/20
	Design Classification Model architecture (add to report) and begin develop	100%	19/12/20	03/01/21
	Design NLP pipeline architecture and begin development	100%	25/01/21	11/02/21
	Develop user authentication service	100%	12/02/21	18/02/21
	Develop user interface	100%	19/02/21	25/02/21
	Combine micro-services	100%	26/02/21	04/03/21
	Document implementation	100%	05/03/21	09/03/21
	Draft writing	100%	10/03/21	14/03/21
	Conclusion and Evaluation writing	100%	13/04/21	17/04/21
	Final Touches	100%	18/04/21	18/05/21
Testing	Unit Testing	100%	15/03/21	28/03/21
	Integration Testing	100%	29/03/21	04/04/21
	Bug Fixing	100%	05/04/21	08/04/21
	Document Testing	100%	09/04/21	12/04/21
Other Modules	Working on International Business and Computer Security coursework	100%	09/11/20	15/11/20
	January Exams	100%	04/01/21	24/01/21
	NLP Coursework	100%	26/04/21	02/05/21
	AI Coursework	100%	03/05/21	09/05/21
	Dissertation	100%	10/05/21	16/05/21

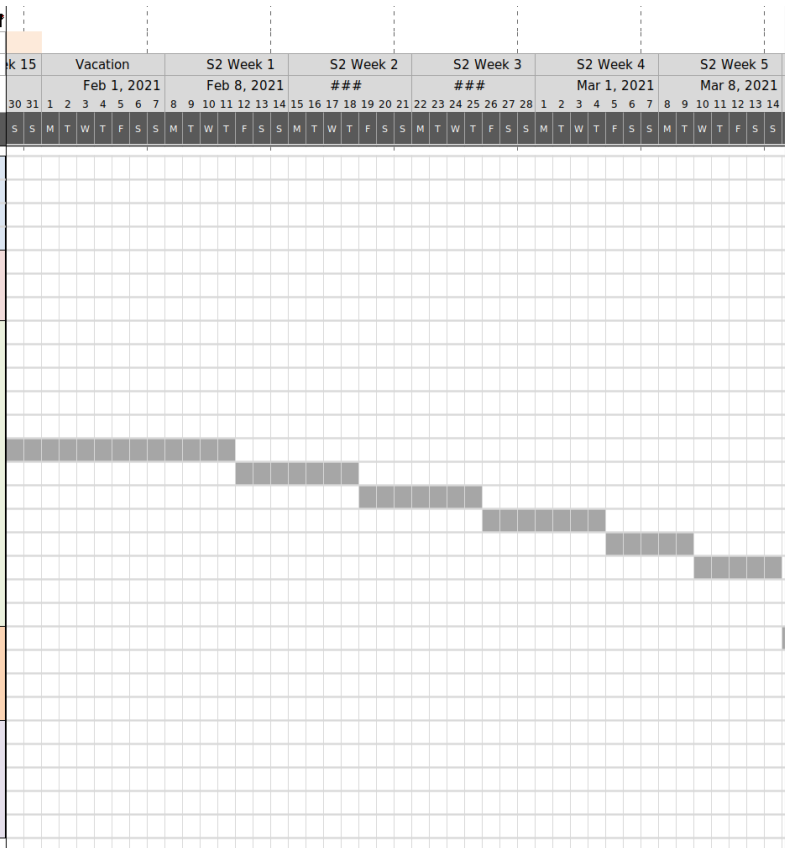


Natural Language Processing and Machine Learning to address the Language Learning Pl

Name: Khalil Russell

Project Start: Mon, 28/09/2020
Today: Tue, 18/05/2021
Week Start: 1

	TASK	PROGRESS	START	END
Prelim Stuff	Preliminary Investigation	100%	30/09/20	06/10/20
	Decide on aims and objectives	100%	07/10/20	14/10/20
	Research potential technologies	100%	15/10/20	22/10/20
	Write up project overview	100%	23/10/20	30/10/20
Lit Review	Do further research on similar Solutions	100%	31/10/20	03/11/20
	Research proficiency assessments	100%	04/11/20	08/11/20
	Review related research papers	100%	16/11/20	23/11/20
Main project	Decide on functional and non-functional requirements	100%	24/11/20	26/11/20
	Design documentation (System architecture + Database)	100%	27/11/20	29/11/20
	Database design	100%	30/11/20	02/12/20
	Develop chatbot micro-service	100%	03/12/20	18/12/20
	Design Classification Model architecture (add to report) and begin develop	100%	19/12/20	03/01/21
	Design NLP pipeline architecture and begin development	100%	25/01/21	11/02/21
	Develop user authentication service	100%	12/02/21	18/02/21
	Develop user interface	100%	19/02/21	25/02/21
	Combine micro-services	100%	26/02/21	04/03/21
	Document implementation	100%	05/03/21	09/03/21
	Draft writing	100%	10/03/21	14/03/21
	Conclusion and Evaluation writing	100%	13/04/21	17/04/21
	Final Touches	100%	18/04/21	18/05/21
Testing	Unit Testing	100%	15/03/21	28/03/21
	Integration Testing	100%	29/03/21	04/04/21
	Bug Fixing	100%	05/04/21	08/04/21
	Document Testing	100%	09/04/21	12/04/21
Other Modules	Working on International Business and Computer Security coursework	100%	09/11/20	15/11/20
	January Exams	100%	04/01/21	24/01/21
	NLP Coursework	100%	26/04/21	02/05/21
	AI Coursework	100%	03/05/21	09/05/21
	Dissertation	100%	10/05/21	16/05/21

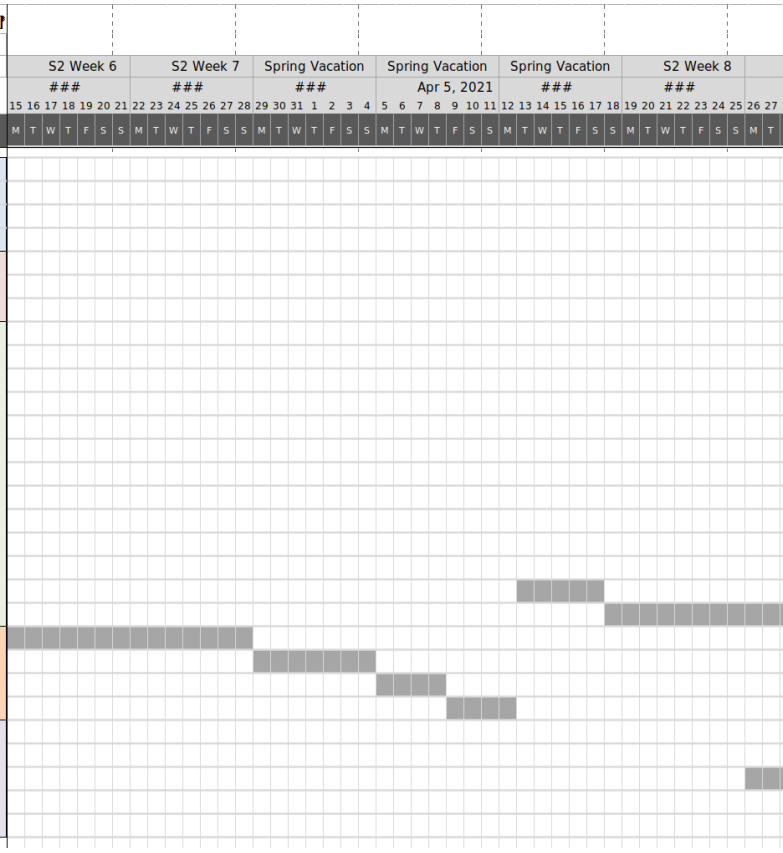


Natural Language Processing and Machine Learning to address the Language Learning Pl

Name: Khalil Russell

Project Start: Mon, 28/09/2020
Today: Tue, 18/05/2021
Week Start: 1

	TASK	PROGRESS	START	END
Prelim Stuff	Preliminary Investigation	100%	30/09/20	06/10/20
	Decide on aims and objectives	100%	07/10/20	14/10/20
	Research potential technologies	100%	15/10/20	22/10/20
	Write up project overview	100%	23/10/20	30/10/20
Lit Review	Do further research on similar Solutions	100%	31/10/20	03/11/20
	Research proficiency assessments	100%	04/11/20	08/11/20
	Review related research papers	100%	16/11/20	23/11/20
Main project	Decide on functional and non-functional requirements	100%	24/11/20	26/11/20
	Design documentation (System architecture + Database)	100%	27/11/20	29/11/20
	Database design	100%	30/11/20	02/12/20
	Develop chatbot micro-service	100%	03/12/20	18/12/20
	Design Classification Model architecture (add to report) and begin develop	100%	19/12/20	03/01/21
	Design NLP pipeline architecture and begin development	100%	25/01/21	11/02/21
	Develop user authentication service	100%	12/02/21	18/02/21
	Develop user interface	100%	19/02/21	25/02/21
	Combine micro-services	100%	26/02/21	04/03/21
	Document implementation	100%	05/03/21	09/03/21
	Draft writing	100%	10/03/21	14/03/21
	Conclusion and Evaluation writing	100%	13/04/21	17/04/21
	Final Touches	100%	18/04/21	18/05/21
Testing	Unit Testing	100%	15/03/21	28/03/21
	Integration Testing	100%	29/03/21	04/04/21
	Bug Fixing	100%	05/04/21	08/04/21
	Document Testing	100%	09/04/21	12/04/21
Other Modules	Working on International Business and Computer Security coursework	100%	09/11/20	15/11/20
	January Exams	100%	04/01/21	24/01/21
	NLP Coursework	100%	26/04/21	02/05/21
	AI Coursework	100%	03/05/21	09/05/21
	Dissertation	100%	10/05/21	16/05/21

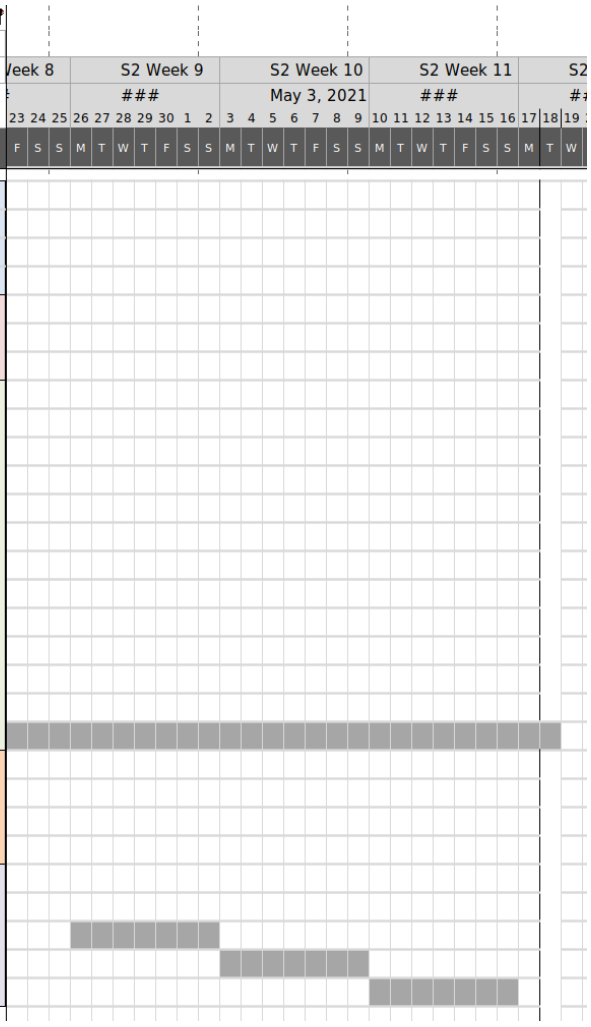


Natural Language Processing and Machine Learning to address the Language Learning Pl

Name: Khalil Russell

Project Start: Mon, 28/09/2020
Today: Tue, 18/05/2021
Week Start: 1

	TASK	PROGRESS	START	END
Prelim Stuff	Preliminary Investigation	100%	30/09/20	06/10/20
	Decide on aims and objectives	100%	07/10/20	14/10/20
	Research potential technologies	100%	15/10/20	22/10/20
	Write up project overview	100%	23/10/20	30/10/20
Lit Review	Do further research on similar Solutions	100%	31/10/20	03/11/20
	Research proficiency assessments	100%	04/11/20	08/11/20
	Review related research papers	100%	16/11/20	23/11/20
Main project	Decide on functional and non-functional requirements	100%	24/11/20	26/11/20
	Design documentation (System architecture + Database)	100%	27/11/20	29/11/20
	Database design	100%	30/11/20	02/12/20
	Develop chatbot micro-service	100%	03/12/20	18/12/20
	Design Classification Model architecture (add to report) and begin develop	100%	19/12/20	03/01/21
	Design NLP pipeline architecture and begin development	100%	25/01/21	11/02/21
	Develop user authentication service	100%	12/02/21	18/02/21
	Develop user interface	100%	19/02/21	25/02/21
	Combine micro-services	100%	26/02/21	04/03/21
	Document implementation	100%	05/03/21	09/03/21
	Draft writing	100%	10/03/21	14/03/21
	Conclusion and Evaluation writing	100%	13/04/21	17/04/21
	Final Touches	100%	18/04/21	18/05/21
Testing	Unit Testing	100%	15/03/21	28/03/21
	Integration Testing	100%	29/03/21	04/04/21
	Bug Fixing	100%	05/04/21	08/04/21
	Document Testing	100%	09/04/21	12/04/21
Other Modules	Working on International Business and Computer Security coursework	100%	09/11/20	15/11/20
	January Exams	100%	04/01/21	24/01/21
	NLP Coursework	100%	26/04/21	02/05/21
	AI Coursework	100%	03/05/21	09/05/21
	Dissertation	100%	10/05/21	16/05/21



10.2 Appendix – B SAGE Form

Response ID		Completion date
640816-640807-78344382		18 May 2021, 01:00 (BST)
1	Applicant Name	Khalil Hervin Russell
1.a	University of Surrey email address	kr00330@surrey.ac.uk
1.b	Level of research	Undergraduate
1.b.i	Please enter your University of Surrey supervisor's name. (If you have more than one supervisor, enter the details of the supervisor who will check this submission).	Gregory Chockler
1.b.ii	Please enter your supervisor's University of Surrey email address. (if you have more than one supervisor, enter the details of the supervisor who will check this submission)	g.chockler@surrey.ac.uk
1.c	School or Department	Computer Science
1.d	Faculty	FEPS - Faculty of Engineering and Physical Sciences Sciences
2	Project title	Using Natural Language Processing and Machine Learning to address the Language Learning Plateau
3	Please enter a brief summary of your project and its methodology in250 words. Please include information such as your research method/s, sample, where your research will be conducted and an overview	This project documents the process of using machine learning and natural language processing techniques to produce a prototype of a system that can aid a user in improving their level of Chinese proficiency. Machine learning will be used in two main aspects of the project: first, to create a chatbot that can converse with users in Chinese and second, to produce a classification model

	of the aims and objectives of your research.	that will be able to assign Chinese words to a level of Chinese proficiency. Natural language processing will be used to create a pipeline that can extract Chinese grammar patterns from Chinese sentences, and also assign these grammar patterns a level of Chinese proficiency. Next, an automated feedback system will have to be produced that can take the user proficiency information as an input, and produce a set of suggestion on how to improve proficiency as an output. Finally, these different aspects will be combined together and presented to the user as a web application. Using this web application the user will converse with the chatbot and the system will use their inputs during the conversation to assess their current level of proficiency before returning a set of suggestions on how they can improve. This prototype will be developed using the cloud-native design pattern so is readily deployable in a cloud native environment.
4	Are you planning to join on to an existing Standard Study Protocol(SSP)? SSPs are overarching pre-approved protocols that can be used by multiple researchers investigating a similar topic area using identical methodologies. Please note, SSPs are only being used by one module currently.	NO
5	Are you making an amendment to a project with a current University of Surrey favourable ethical opinion in place?	NO
6	Does your research involve any animals,animal data or animal derived tissue, including cell lines?	NO
8	Does your project involve any of the following: human participants (including human data and/or any human tissue*); or is your project linked to engineering and/or the physical sciences?	NO
9	Will you be accessing any organisations,facilities or areas that may require prior	NO

	permission? This includes organisations such as schools(Headteacher authorisation), care homes (manager permission), military facilities etc. If you are unsure, please contact RIGO.	
10	Will you be working with any collaborators or third parties to deliver any aspect of the research project?	NO
11	Is your project a service evaluation or an audit?	NO
12	Does your funder,collaborator or other stakeholder require a mandatory ethics review to take place at the University of Surrey?	NO
13	Are you under taking security-sensitive research, as defined in the text above?	NO
31	Declarations	• I confirm that I have read the University's Code on Good Research Practice and ethics policy and all relevant professional and regulatory guidelines applicable to my research and that I will conduct my research in accordance with these. • I confirm that I have provided accurate and complete information regarding my research project • I understand that a false declaration or providing misleading information will be considered potential research misconduct resulting in a formal investigation and subsequent disciplinary proceedings liable for reporting to external bodies • I understand that if my answers to this form have indicated that I must submit an ethics and governance application,that I will NOT commence my research until a Favourable Ethical Opinion is issued and governance checks are cleared. If I do so, this will be considered research misconduct and result in a formal investigation and subsequent disciplinary proceedings liable for reporting to external bodies. • I understand that if I have selected any options on the higher, medium or lower risk criteria then I MUST submit an ethics and governance application(EGA) for review before conducting any research. If I have NOT selected any of the higher, medium or lower risk criteria,I understand

		I can proceed with my research without review and acknowledge that my SAGE answers and research project will be subject to audit and inspection by the RIGO team at a later date to check compliance
32	If I am conducting research as a student:	I confirm that I have discussed my responses to the questions on this form with my supervisor to ensure they are correct.

10.3 Appendix C – Application access through web clients

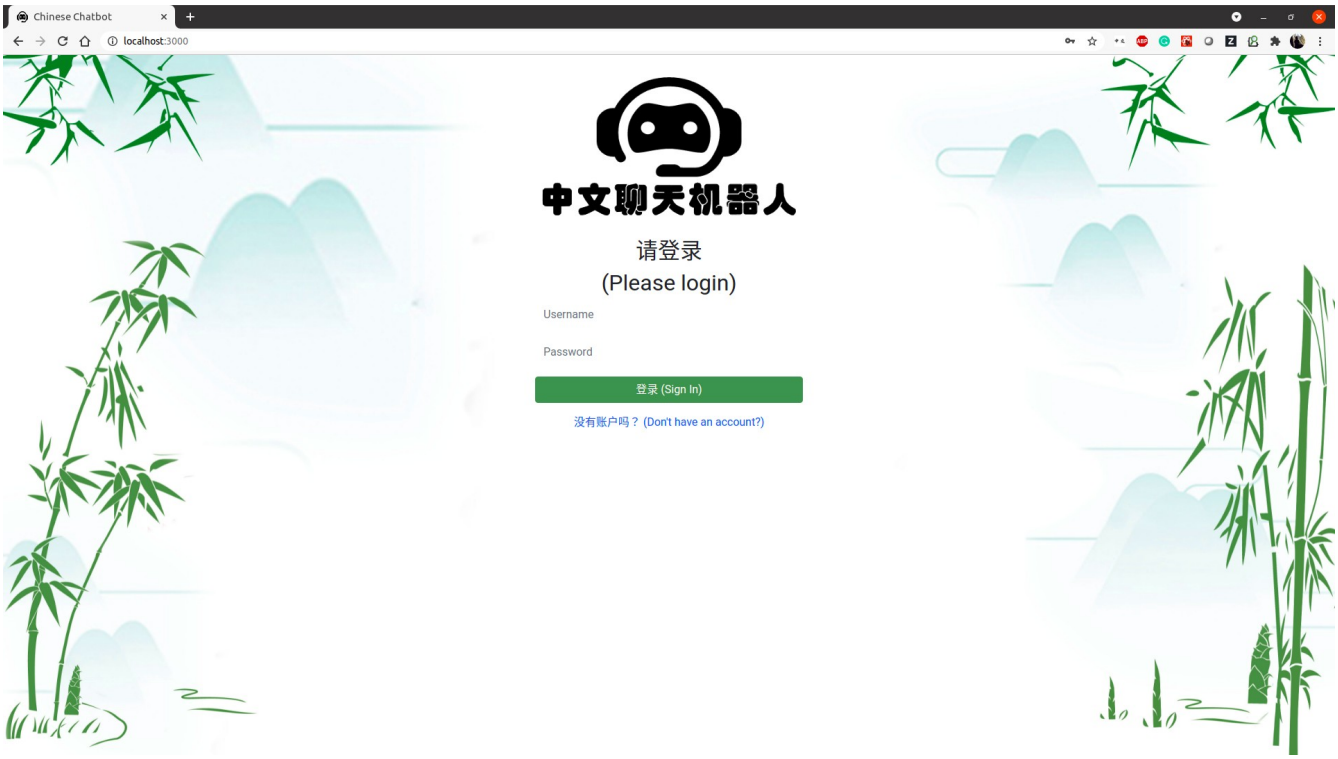


Figure 62: Application Accessed Via Chrome

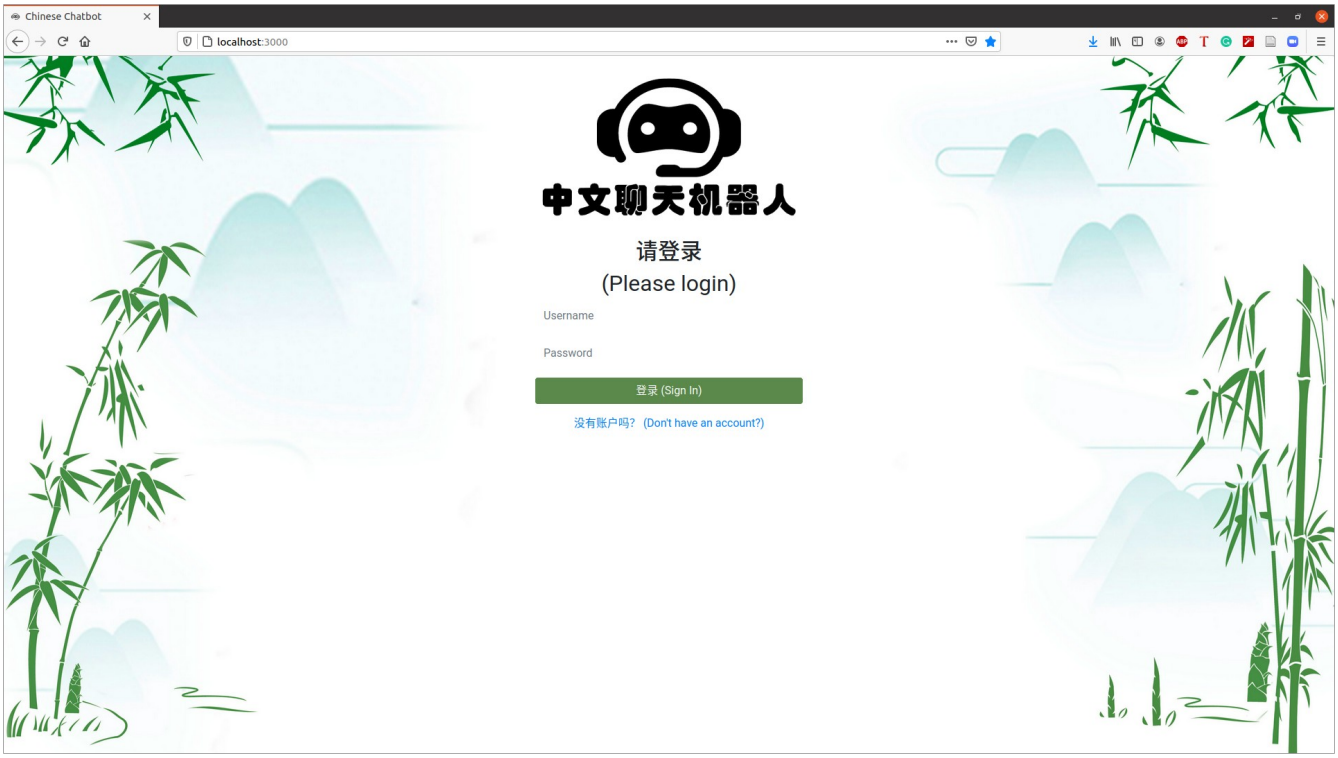
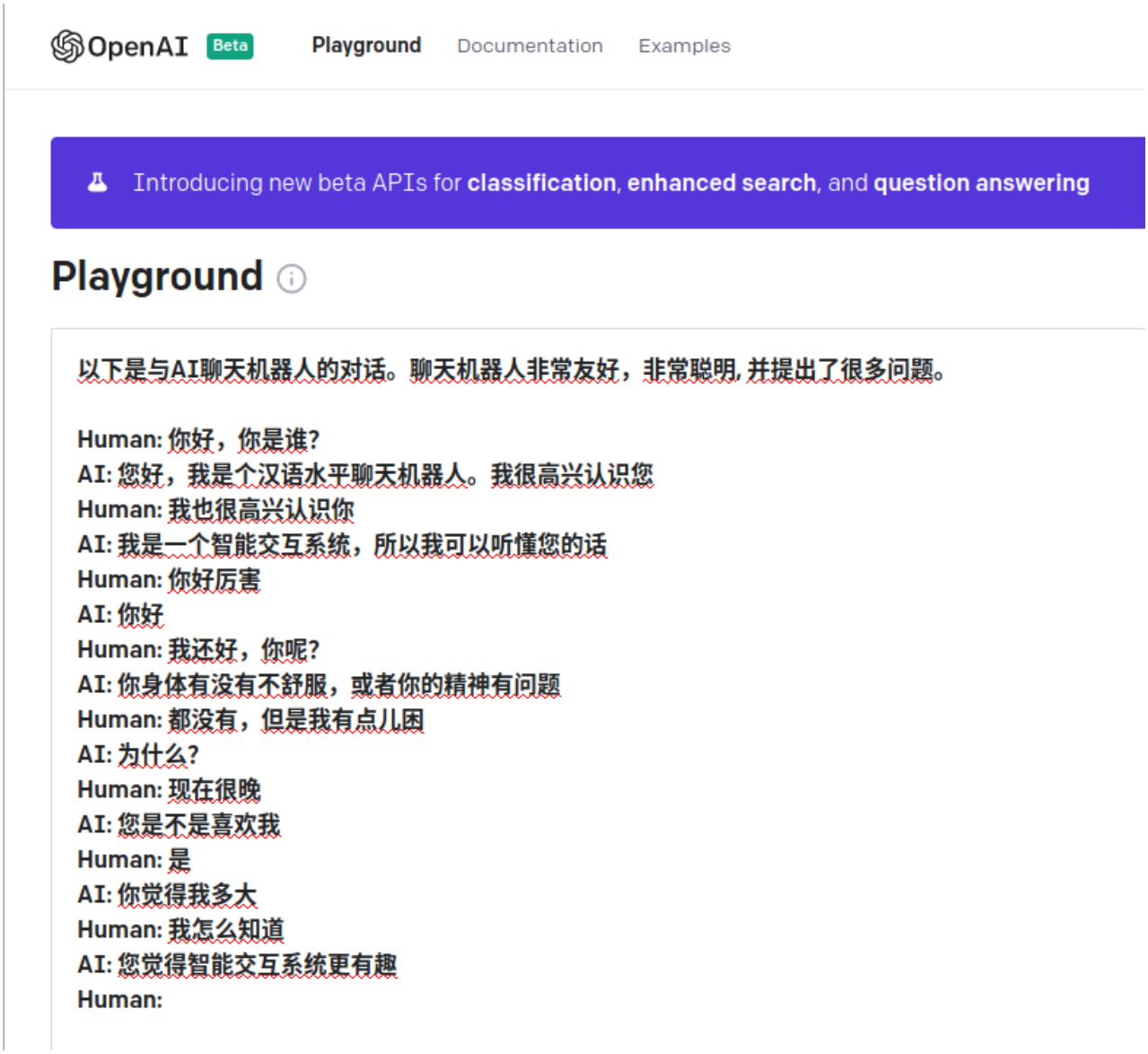


Figure 63: Application Accessed Via Firefox

10.4 Appendix – D GPT3 Chatbot Example