**CA400 Final Year Project**

**Technical Guide**

**DCU Campus Chatbot**

**Sam Wood - 16476066**

**Mark Agnew - 16315811**

**Project Supervisor: Gareth Jones**

**Table of Contents**

[**1. Introduction**](#_9gob9swvfkxq) **3**

[1.1 Abstract](#_rqzvp3mjld0) 3

1.2 Glossary 4

[1.3 Research](#_wgweake6na1b) 4

[**2. System Architecture**](#_vqn5mhnv7y3w) **6**

[2.1 High-level overview](#_1duej1xqwr93) 6

2.2 What is AIML 6

[2.3 Interacting with the App](#_gdi429tqy3u) 7

[2.4 In conversation with the Chatbot (AIML Files)](#_zijwy9kb05hl) 7

[**3. High level design**](#_fuusm2kfj1bj) **9**

[3.1 Sequence diagram](#_ge5cjy4owo3o) 9

[3.2 Context diagram](#_u3vnz62vej5g) 10

3.3 Data flow diagram 11

[**4. Technical details and Implementation**](#_qgfw1t25vi2b) **12**

[4.1 Artificial Intelligence Markup Language](#_f54g7ptbch3) 12

[4.2 Creating the UI (using HTML,CSS and JS via flask to render )](#_lip3c16ev2ia) 13

[4.3 Integrating the front and back-end (Python and JS)](#_nrw9x2mhme4t) 14

[**5A. Problems and resolution**](#_ti5guyel0kmo) **15**

[5.1 Traceback Function](#_jyxl6af4zv6s) 15

[5.2 Spell checker system](#_e74n6vo49gtv) 16

[5.3 Images](#_if6ecbop24vm) 17

[5.4 New technologies](#_dvyptt2m4ltf) 18

[5.5 Covid-19 restrictions](#_7a2buyeebjh5) 18

[**5B. Smaller challenges**](#_r8tuz0btq2ah) **19**

[5.6 Building and Lecture room Directions](#_jkepggh544fh) 19

[5.7 Splitting messages](#_1yx4jnxu8wqf) 20

[5.8 Maintain intelligence](#_ohycp5osz164) 21

[5.9 Let the user know how to use the application](#_rwuanis5xrch) 22

[**6. Conclusion**](#_gm8onb6xwk5c) **22**

[6.1 Results](#_40mzcg4y2n21) 23

[6.2 Future Work](#_ozqhylglz4m2) 24

# 

# 1. Introduction

## 1.1 Abstract

Our project is an omni-capable web app that is designed to accommodate users who are new and unfamiliar with the Dublin City University Glasnevin campus. The primary functions of the application are to decipher the codes used to reference the various buildings and locations on campus, provide directions to the various locations and also lecture rooms, provide images of the buildings and also to answer frequently asked questions about the University. While giving the user access to these services, we have also made it a priority for the user interface to be customizable to the users preference. To do this, we have enabled an options menu which allows them to change the font, it’s size and the colours of the message environment. Unfortunately, due to the Covid-19 outbreak and the fact we were recommended to not be on the DCU campus for much of our second semester we were unable to collect much of our data i.e. campus room locations.

Given that neither partner has designed a full web application before this project, we felt it was appropriate to create this service in the form of a website and to make the website also accessible from smart phones. This way our app is accessible from laptops, desktops and handheld devices, similar to many of today’s popular applications such as Facebook, Twitter and Instagram. Given the popularity of these services and that we hope to be entering the workplace later this year, we assumed we would benefit from making a service that is available on more than one digital channel. It has also been a goal of ours to make the app simple and as intuitive to use as possible and as a result, have created an application that is useful to anyone that uses it.

With regard to our motivation to create this application, it came from when we were new to the DCU campus in the first year of college. On numerous occasions students would end up at the incorrect rooms and sometimes incorrect buildings for lectures due to how the campus map is constructed. For example, ‘HG’ is very commonly misinterpreted as the Henry Grattan Building even though it is the bottom floor of the Nursing Building. For reasons like this, we feel our application would be a good contribution to the many things DCU has to offer, to improve the everyday life of it’s students and visitors.

This document outlines our systems architecture, high-level design, technical details of our project, problems and resolutions we encountered and how to install our project on a device. The instructions on how to use our project to someone unfamiliar with it are kept in the user manual and our testing document contains all of the information on how we tested our application before submission.

## 1.2 Glossary

*Dialogflow -* This is a Google-owned developer of human-computer interaction technology based on natural language conversations.

*Entities -* This is a property in Dialogflow which can be used to answer which can be used to answer the request from the user. The entity will usually be a keyword within the request like the name, date or location.

*HTML* - Hypertext markup language

*AIML* - Artificial intelligence markup language

*IDE* - Integrated development environment

*API* - Application Programming Interface

*UI -* User Interface

*Flask - A web microframework for building web applications with python*

*Fuzzy wuzzy - A python library for string comparison*

## 1.3 Research

Upon completing our functional specification in november, we had a collection of technical requirements that needed our attention;

* Functional User Interface
* User start up
* The ability to parse queries and respond effectively

We also had to decide how it was all going to be programmed. When we knew we were going to build a web app it was obvious we would be using technologies like HTML, Javascript and CSS, given our experience with them in our modules at the beginning of our degree. After investigating how chatbots are commercially designed, it was apparent that our project would entail designing a neural network or using technologies associated with the field.

One of the questions that lingered over our heads was if we would design our own Neural Network from scratch or use one of the API’s for creating chatbots such as Google’s Dialogflow (a natural language understanding platform used to design and integrate a conversational user interface in mobile, web apps etc) . Our supervisor pointed out that it would be much more difficult to get a simple working chatbot if we programmed our own neural network and would be very hard to get a very capable chatbot due to the lack of data needed. It occurred to us that Artificial Intelligence Markup Language (an XML language used to design chatbots) or DialogFlow would have been the way to go.

Following much research on Dialogflow and making smaller chatbots that would resemble some of our features, we realised that if we went with this API there would not be much code produced by us at all. We knew that from a difficulty perspective for the project that this was a problem. For this reason we decided to use AIML as we would be producing our own code instead of it being automatically generated and there would be a challenge in learning a new language. With the use of youtube tutorials and various websites we learned the basics of AIML and were able to work well with the language. Usually in application design, the system architecture would be the first thing addressed. In our case, we designed this project from available technologies. This research led to our diagram in figure 2.1.

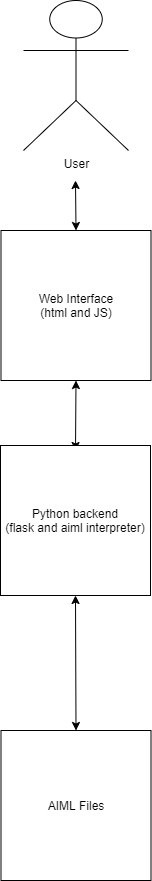
Another aspect we chose to implement in the chatbot that we came across was the Gestalt principle. These are laws and rules that describe how the human eye perceives visual elements. We felt it was important to keep the layout nice for the user i.e. not too much reading and for it to be aesthetically pleasing.

.

# 

# 2. System Architecture

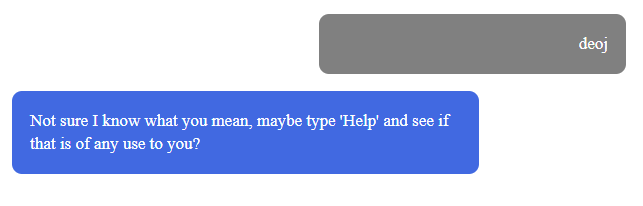
## **2.1 High-level overview**



*Fig 2.1 A representation of our system as a whole including the user*

**2.2 What is AIML?**

AIML is an XML based markup language meant to create artificial intelligent applications. AIML makes it possible to create human interfaces while keeping the implementation simple to program, easy to understand and highly maintainable. The chatbots queries and responses are handled by this section of our code. The user sends queries/messages using the chatbot interface and if the query matches an entity in the files, the response is forwarded to the user. Otherwise, the default response is sent.

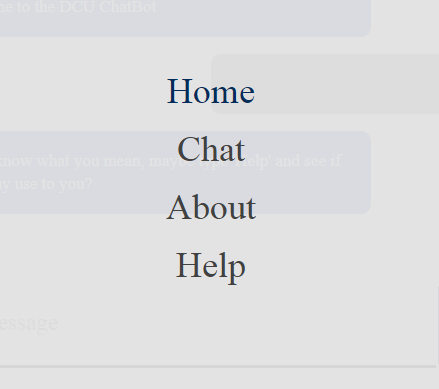
****

*Fig 2.2*

The above figure is an example of the default response when a query isn’t recognised.

## 2.3 Interacting with the App

When our web app is launched the user is brought to our home screen where the user is shown how to start the chatbot and also minor details about the help and about pages. The menu button at the top corner of this page allows the user to navigate between the pages for ‘Home’, ‘Chat’, ‘About’ and ‘Contact’.

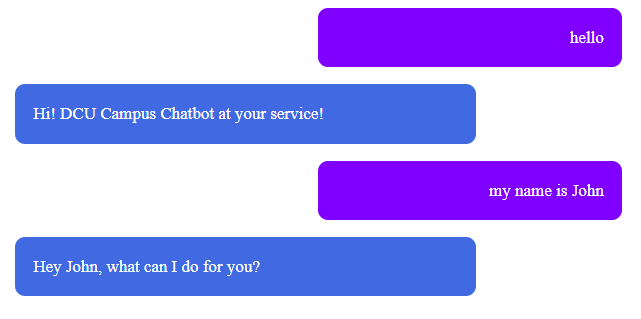


*Fig 2.3 The chatbots home menu*

## 2.4 In conversation with the Chatbot (AIML Files)

We used Artificial intelligence markup language (AIML) for implementing our chatbots functionality. There are a number of AIML files in our repo. Each file is named according to its function. When the user selects the chat option they are greeted by the chatbot and are free to make queries to the chatbot. The interface works as a tunnel for the queries to be sent by the user, parsed and thus a response is given.

When designing the application we thought it might have seemed bland and not as effective if our chatbot was only able to deal with the functions we specified. For this reason we thought it necessary to let our app have basic chat qualities like being able to respond to a user using a greeting phrase and introducing themselves.



*Fig 2.4 Someone introducing themselves*

Given that our app will mainly deal with college students and it would have seemed , we felt it was appropriate to make the chatbot capable to respond to colloquial phrases that students may use like ‘how are you ?’ and ‘how are things’ etc. The functions file handles queries about what the app does outside of the help function. An example would be ‘can you help me with directions ?’ and ‘Will you help me with building codes?’

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

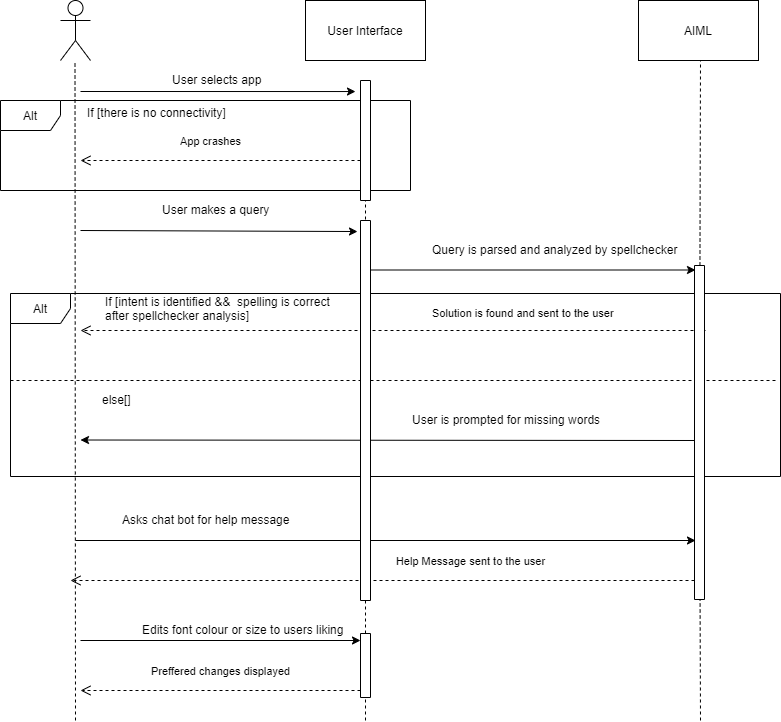
# 

# 

# 

## 3. High level design

## 3.1 Sequence diagram

****

*Fig 3.1*

The above is a generalisation of our app in the form of a sequence diagram. The diagram portrays the order of interactions that take place between the user and the chatbot. The user interface and aiml subsystems are represented by the square boxes, the arrows are the passing of information between the subsystems and the dotted lines are the timeline of events.

## 

## 

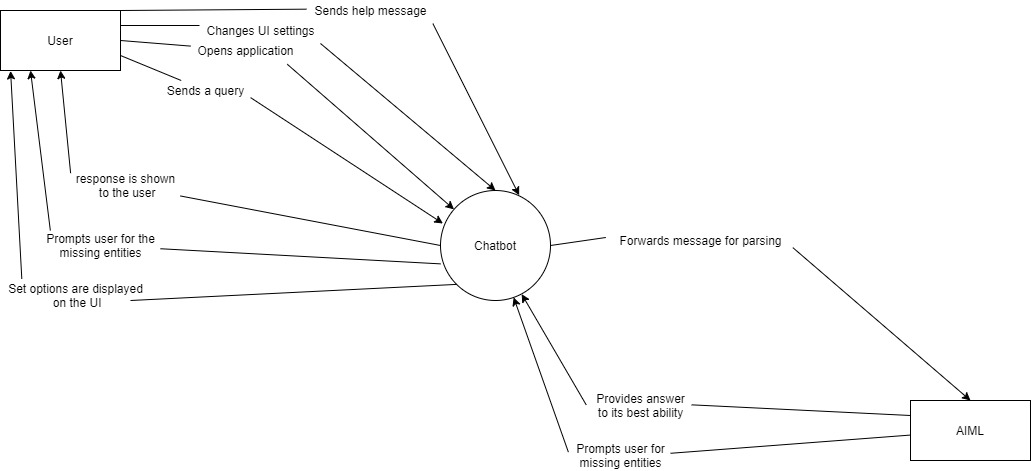
## 

## 

## 

## 

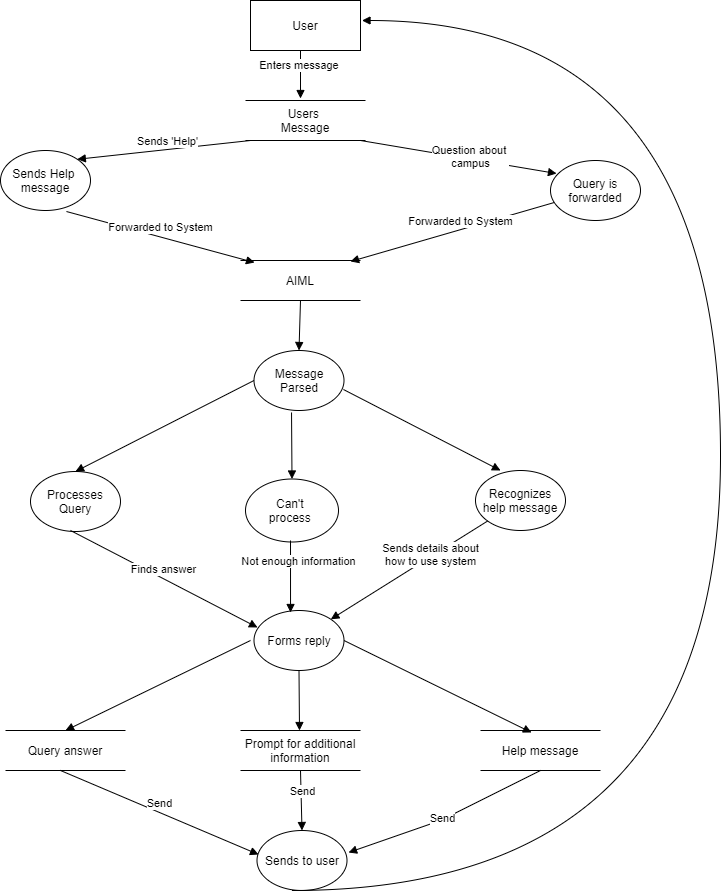
## 3.2 Context diagram

****

*Fig 3.2 Context diagram*

Above is our application viewed as one huge process, the chatbot is represented as the circle in the centre. The external entities in this case are the user and the AIML code which are represented by rectangles. Each action that the user takes and what it makes the chatbot do is represented by the arrows.

**3.3 Data flow diagram**

****

*Fig 3.3 Data flow diagram*

This is the DCU chatbots data flow diagram. The arrows represent the flow of our data from our entities to the various data stores. Once the system has responded to the user the process will repeat itself. A conversation with the chatbot is a number of iterations.

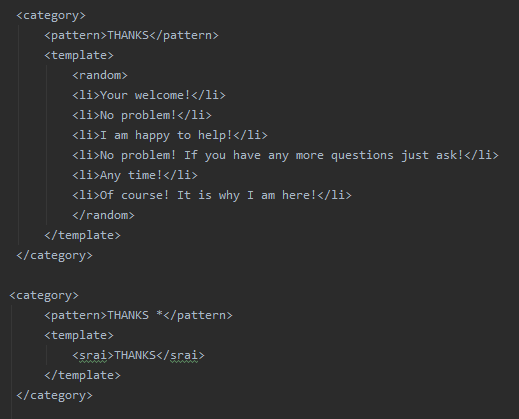
The python backend both renders the webpages for the user using flask, but also has a few other functions. The first of these is to retrieve the users query from the chat webpage, it does this through flask by issuing a get request for the users message. Once it gets this message it then performs a spell check on it to correct any possible typos, that corrected message is then sent passed to the library used to get a response from our AIML files to then return to the user.

# 4. Technical details and Implementation

## 4.1 Artificial Intelligence Markup Language

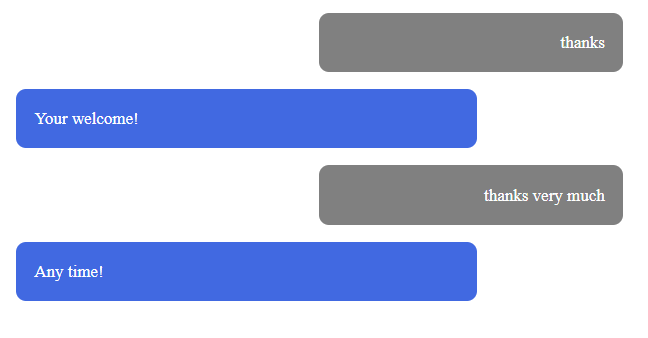
AIML allows us to create human interfaces while the implementation is simple to program, easy to understand and highly maintainable. For this reason we thought it would have been appropriate to use this technology in designing our chatbot.

Our AIML files are learned in the “std-startup.xml” file which is called in the chat.py file. The most basic function used in AIML is with the “category” tag which contains the “pattern” and the “template” and all together allow a query to return a response.



*Fig 4.1*

The above diagram shows the use of the “category”, “pattern”, “template”, “random” and wildcard tags. In the first block of code, the code will take in the “thanks” input and the chatbot will respond to one of the phrases in the “<li>” tags at random. This makes the chatbot seem more human by not just saying your welcome each and every time. The second code segment uses the wild card tag. The “\*” represents anything to follow the “thanks” statement can be accepted. And the <srai> tag means to treat the pattern the same as the response in the <srai> tag.



*Fig 4.2*

The above is a snippet of the code from the previous page in action. This is just an example of how the AIML tags are used in our project. The wildcard tag was particularly useful for allowing the bot to take in additional phrases to trigger the same function.

## 4.2 Creating the UI (using HTML,CSS and JS via flask to render )

The next step we took in implementing our design was to create the webpages for the application itself. This section provided many challenges as there are quite a few aspects we needed to get correct. Firstly we had to design the website to be user friendly and accessible to ensure that all users could engage with our application.

We decided to keep it simplistic with 4 different web pages, The landing page would allow the user to go straight to the chatbot and briefly let the user know what information is available on the other pages. The colour scheme of the pages is made to match that of the DCU logo’s blue, which along with white provides a high level of contrast making text easier to read. The menus are also simple popouts in nature and are created using a mix of all three languages in the heading of this section, this is to make the options and menu easy to navigate. To make the site more accessible to those with visual impairments we also allowed them to change the background colour of the text boxes in the options menu along with the font sizes,font style and user/chatbot message labels.

The second challenge was learning how to design all these features as we have only done 1/2 modules involving these languages so our experience was limited so there was a lot of learning involved and trial and error to get the positioning of elements correct. There was also learning of how to use flask to create templates so the navigation bar code was not duplicated in every html file, flask would also later allow us to integrate our code through python.

## 4.3 Integrating the front and back-end (Python and JS)

In order to bring the back and front-end of our code together we used python, this also allowed us to implement extra functionality in the form of a spellchecker (using the autocorrect and fuzzywuzzy libraries mentioned in the user manual) and to provide a traceback functionality for any mistaken inputs.

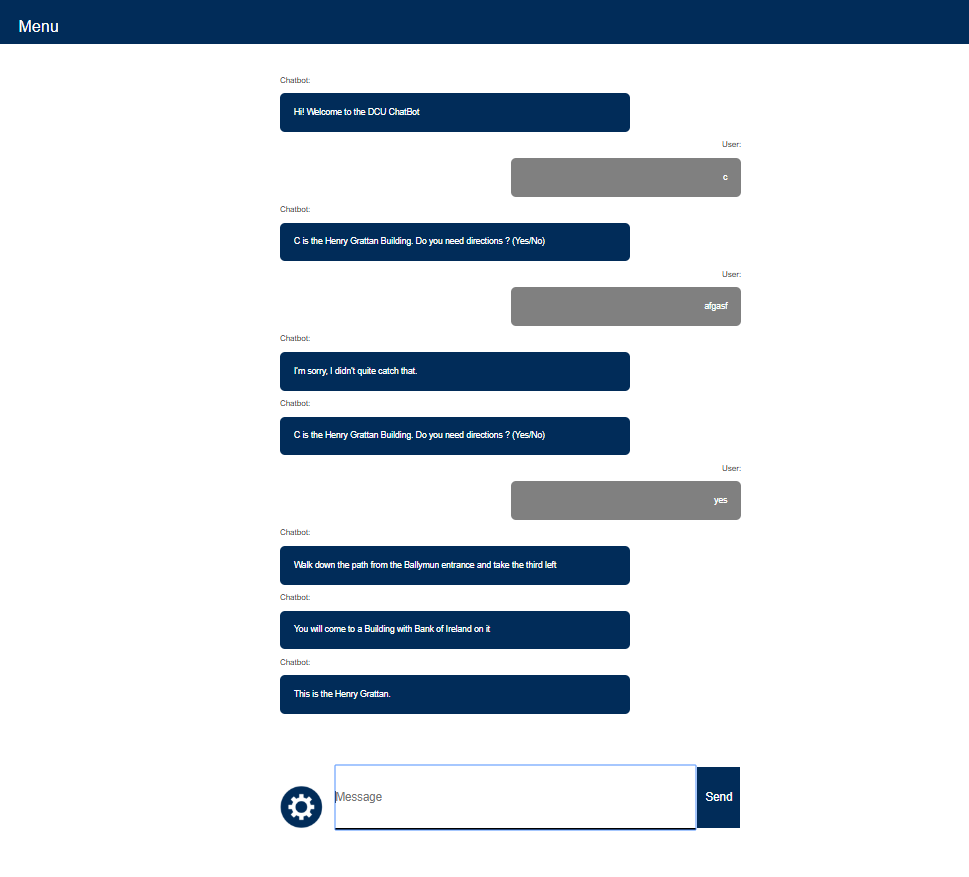
To interact with the aiml code we wrote we used the python aiml library through which we can load up our aiml files, send a message to and retrieve a response from. In order to get the message from the user at the front-end we used flask to render/run the html files then javascript along with jquery to generate a get request when needed to retrieve the users message then pass it to the aiml interpreter in python.

The javascript is used to provide a response for the get request and also to retrieve the response from the aiml interpreter in the python code and display it for the user to see.

# 5A. Problems and resolution

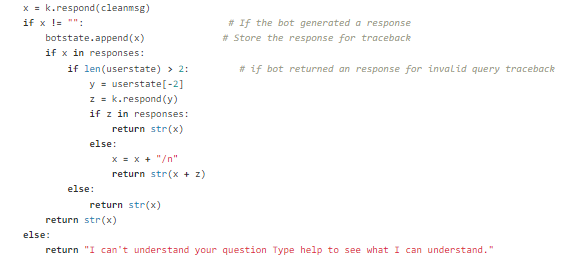
## 5.1 Traceback Function

It is no surprise that people make errors when typing an SMS message. The features of our application are triggered when the query sent by the user matches what is written in our AIML code. Also, many of our commands are triggered by following up to a reply that has already been sent to the user, using an AIML “that” tag. Hence or otherwise if the user sends the triggering message to the reply but spells the response incorrectly, they will have to essentially begin the conversation again. We fixed this issue with the traceback function. See an example of the scenario below.

.

*Fig 5.1*

As you can see, the user has been saved the trouble of asking the bot for the meaning of ‘C’ a second time because of their misspelling. This was done using lists for the botstate and userstate so if there is a query that is misspelled and not resolved by the spellchecker, not much effort is required by the user. It also contributes to the ‘realness’ of the chatbot.

.

*Fig 5.2 Get bot response function*

The “userstate” and “botstate” functions are initialized as empty lists in our “Chat.py” file above this function in the source code.

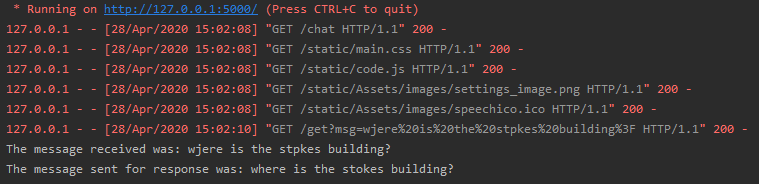
## 5.2 Spell checker system

In order to make the bot more user-friendly and functional we added a function to correct any possible errors in the users message to allow the system to return a correct response.

At the start of implementing this feature we tried two different libraries (autocorrect and pyspellchecker) that correct general spelling, after testing both we found they had the same issue when it came to correcting spellings for names of buildings on campus as they are not in either libraries dictionaries. To resolve this issue we then used another library in order to compare (using Levenshtein Distance to calculate the differences between sequences in this case words) a word corrected by one of the original libraries to a list of building names,words and commands that are unique to our application. If the correction has a greater than or equal to 67% chance of being one of the words in our list we swap it out for our word (67% was the limit as it was needed in some cases like the misspelling of “union”).

Then while carrying out unit testing on our project we noticed that if a word was misspelled and we were using the pyspellchecker library it was slowing down the process by a large amount, switching to the autocorrect library lowered this time to an acceptable level.

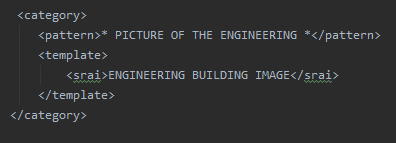
Below is an example of a message with incorrect spelling and the bots response and another image showing how that it is corrected behind the scene with a test line showing the message received and the message sent to retrieve a response.



*Fig 5.3*

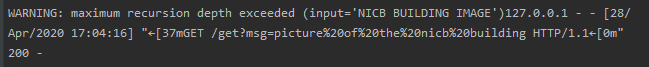
## 5.3 Images

Initially, our images of the buildings would be retrieved with the title of the building followed by the word ‘image’. We noticed that it isn’t a very intuitive feature to have, i.e. the user wouldn’t not know to ask for them in that way unless told to. For this reason we decided to employ numerous phrases for asking the bot for the pictures. Like much of the rest of our code, we used the recursion through the AIML wildcard tag.



*Fig 5.4*

This means that the string between the pattern tag is accepted the same as the string between the srai tags as you can see in the image above. Since this tag is recursive it contributes to the smoothness of our code. After we had included numerous phrases we received a maximum recursion error as you can see in the image below.

****

*Fig 5.5*

This happened with a number of the image retrieval sections in our code. To overcome this we had to remove some of the wildcard tags. Although it meant there would be less recursion in our code and our overall application might not run as smoothly, it had to be done so the errors would disappear and our application would work fully.

Another problem with our images feature was how the questions could have been phrased. Before we had included the images feature, our ‘building directions’ file was picking up the name of the building and recognizing it as a directions query. Here is an example of the problem in the figure below.



*Fig 5.6*

Usually we would've just used the ‘\*’ following the building name but this had to change as the building name was going to be in the image query. There was a lot of clashing with the commands and as a result we had to consider many ways the user could ask for the image and program it for each building and program it. This led to our images AIML file being much longer than the rest but had to be done so the application would work how we had specified.

## 5.4 New technologies

We learned a great deal when designing this project but another challenge that is worth mentioning is the new technologies in our project. After designing several small chatbots that would have been minor versions of what our full product would have been, we grew less and less inclined to use DialogFlow as our main tool for the project. We thought from our examiners perspective that this system that was just generating code for us while we basically ‘filled in the blanks’ had potential to reduce the score of our final mark. After much consideration of this, AIML was looking more and more like our solution.

After much more studying on the topic and looking at various websites and watching youtube videos, AIML grew on us as a technology for us to work with. Having very little experience with XML between the two of us (very briefly in the third year project) we thought that it may have been challenging, but it was a learning experience that we enjoyed nonetheless.

We also used languages that we had some experience in from first and second year of our degree (such as HTML,CSS and JavaScript) but had to learn and develop on those things to create our project, such as the javascript use to allow the messages be entered on the website then passed to the python back-end code which was an entirely new learning experience. HTML and CSS also proved difficult to get the positioning and look of elements correct so that they would be visually pleasing to the user and easy to use, it took a lot of trial and error and some user testing to make sure that this ended up looking correct.

## 

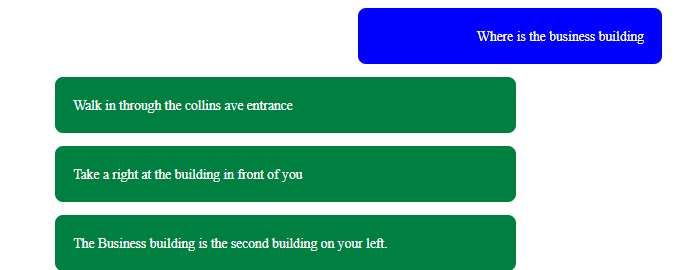
## 5.5 Covid-19 restrictions

Clearly, a lot of the data we require for our project would have been acquired by going around the campus and collecting it ourselves i.e. directions to rooms and buildings and pictures of the buildings. Unfortunately, due to the Covid-19 outbreak and the fact we were recommended to not be on the DCU campus for the second half of semester 2 this year. Thankfully using the campus map, we were able to program a substantial amount of building directions. To overcome not being on campus to take the pictures, we made use of google images and google earth by taking screenshots. As a result, we have images for a number of buildings. Hence we have shown that we can enable this feature. We were also not able to collect directions to all of the lecture rooms but had already coded directions to rooms in the computing, business and nursing building. Likewise we have shown that the feature can be completed.

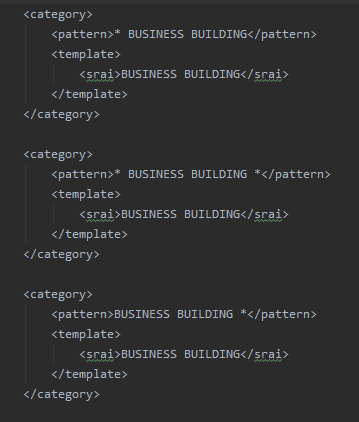
# 5B. Smaller challenges

## 5.6 Building and Lecture room Directions

Following our unit testing of the bot responses we realised that we were limited in ways the user could ask for directions to the lecture halls and buildings. For example, if the user asked the full sentence ‘Can you tell me where the business building is?’ The bot would respond that it cannot understand the query. This meant for a significant amount of additions to our aiml files for building directions and lecture room directions. Following the extra lines of codes that had been added after this realisation, the user could ask for the directions in any way possible and the answer would be provided. See screenshots below.



*Fig 5.7*

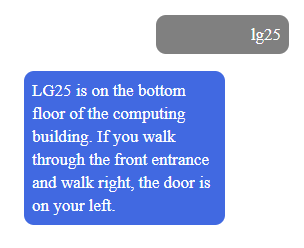
**

*Fig 5.8*

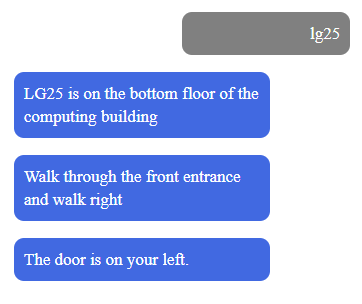
The ‘\*’ symbols mean to accept any phrases but not an empty string. As a result the user can ask any way he/she would like for the building’s and lecture rooms and the bot would understand the query perfectly.

## 5.7 Splitting messages

Following several meetings with our supervisor, it was pointed out to us that some of our bots responses were long and in the form of a block of text when they didn’t need to be. These blocks of text were an unattractive feature in our app and we had to deal with it somehow. Our solution was to split the messages up into more than one message, and shorten our sentences.



*Fig 5.9 Our responses before the split*

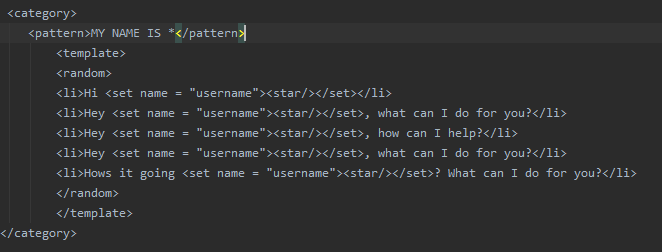
****

*Fig 5.10 responses after the split*

Clearly, this new and improved bullet pointed format is nicer for the user and improves the layout of the chat environment.

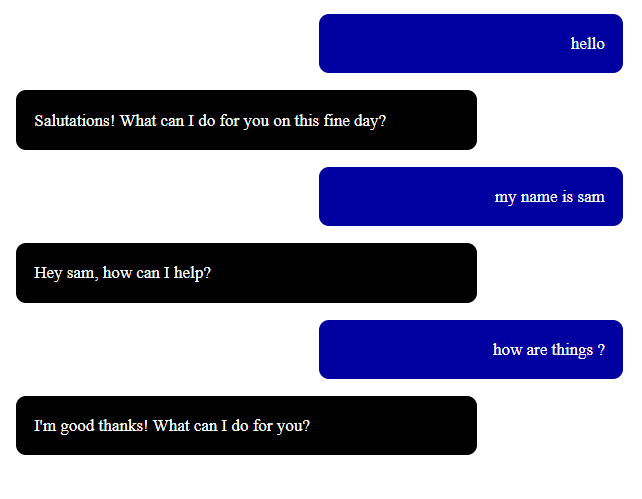
## 5.8 Maintain intelligence

As important as it was for our product to be able to deal with issues like deciphering building codes and to give directions to buildings and rooms. We couldn’t help but feel that a system that only dealt with these features would seem incomplete or boring. To deal with this issue we expanded our basic chat AIML file. We gave our chatbot the ability to be able to respond to colloquial phrases and to respond to the user introducing themselves. The bot was then able to respond and remember the user’s name.

****

*Fig 5.11*

Above is how the chatbot takes in the user introducing themselves and is able to respond with their name.

****

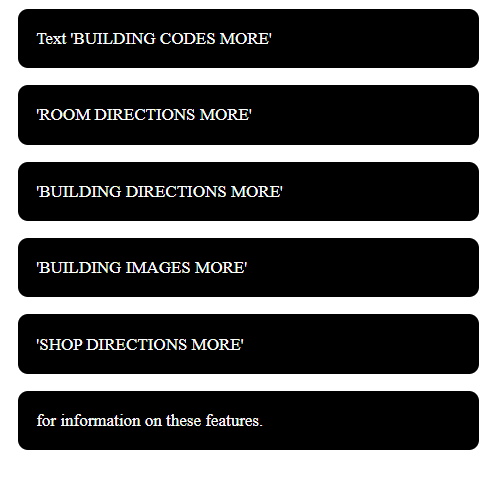
*Fig 5.12*

Above is a snippet from a conversation showing that the user can introduce themselves and include a commonly used phrase and the app will know how to respond. This is an important aspect as a huge part of a chatbot is to be able to let it seem the user is talking to a real person.

## 

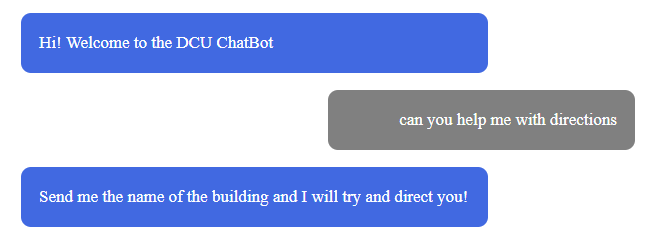
## 5.9 Let the user know how to use the application

This area was more relevant than we had initially anticipated but as the project went on we realised that the user will not not know how to use the application if there is not some kind of help feature. For this reason the user can gain information on all of the main features by sending the right message. The user needs only type the word help and then they are given a list of commands that will describe each feature in the right amount of detail.

****

*Fig 5.13*

In addition to these results we have also implemented a functions AIML file which contains commands that allow the user to ask about the functions of the chatbot i.e phrases like ‘Can you help me with lecture rooms’ and ‘can you help me with images’ all provide valid information to the user.

****

*Fig 5.14*

Above is an example of how the user might learn about how to use the chatbot without the use of a user manual or something similar.

# 

# 

# 6. Conclusion

In this section we will discuss our final web application and how useful it is at its current point of development, we will also show the possibilities of this application going forward noting how it could be further improved and iterated upon.

## 6.1 Results

Based on both our research and user testing through a survey and follow up interviews it is clear that there is a clear use for an application such as this. From our survey we found out that over 70% found it personally useful and that 100% thought that it would be somewhat or very useful for those new to/visiting DCU Campus (i.e First year students, Erasmus students and guest speakers). Our research also showed that no major universities or other 3rd level education institutes have a similar application for use by their student body showing there may also be a market for this application.

## 6.2 Future Work

Throughout the development of this project we came across many ideas that could be used to further develop our application. The first of which was to change the back-end to something more towards machine learning, this could not be done initially as you would need to gather data from a chatbot like ours in order to train a machine learning algorithm. This would allow for better natural language processing and possibly more tailored responses.

Next we thought a simple but useful addition would be to include different languages that could be used with the chatbot so it could be used by erasmus students more easily. From our user testing there were a few highlighted directions the project could go, some of these include:

* Creating a standalone app.
* Directions using gps and map.
* Give updates on what’s happening with clubs and societies.
* Allow the bot to respond with videos

These are just some of the many ways this could be furthered as an application to be more commercially viable for universities to implement. This shows that this is a promising project with a lot of potential to become widespread.