National College of Ireland

BSc in Computing

2016/2017

Karl Lyons

X13452508

Karl.Lyons@student.ncirl.ie

Intelligent Chatbot

Technical Report

Table of Contents

Executive Summary

Intelligent Chatbot is a project that explores what can be accomplished with AI agents in the world we live in today. Alan Turing (1912-1954), known as the father of modern computing, developed the idea of artificial intelligence. Perhaps most famously known for breaking the Enigma code used by the Germans in WWII, which was widely regarded as unbreakable, he lay the foundations for the development of computer science explored what is possible using electronic computers and artificial intelligence. In his 1950 paper on Computing Machinery and Intelligence, he asks the question; "Can machines think?". He created a test, with the fitting name of the Turing test, whereby a machine is tested on its ability to exhibit intelligent behaviour equivalent to, or indistinguishable from, that of a human. Throughout the paper he delves deeper into this idea and I came across an interesting quote. To quote directly from Turing, "I believe that in about fifty years' time it will be possible to programme computers, with a storage capacity of about 10^9, to make them play the imitation game so well that an average interrogator will not have more than 70 per cent chance of making the right identification after five minutes of questioning."[Turing, 1950].

He wasn't too far off. In June 2014, a computer program called Eugene, which simulated a 13 year old Ukrainian boy was said to have successfully passed the Turing test at an event held at Reading University's School of Systems Engineering. Some artificial intelligence experts refuse to recognize this as a legitimate victory but it is interesting to note how close Turing's prediction was.[BBC tech, 2014]

Messaging applications are the #1 most used applications in the world at the moment and some businesses are already taking advantage of this and developing chatbots of their own to complement their social media presence. With messaging applications taking up the majority of a users time, it makes sense for businesses to capitalise on this. Bots are becoming smarter and faster and will soon become easier to use than a website or app. A recent inquiry into the area of chatbots indicated that 180 bot-related

companies have attracted $24 billion in funding towards the development of AI driven bot applications. [Gatti, Venturebeat, 2017]

Microsoft CEO Satya Nadella was quoted saying “Bots are the new apps”.[Cava, USA Today, 2016] With Microsoft set on exploring deeper into the world of chatbots and accelerating the innovation and sophistication of chatbots, I’ve found this project both interesting and exciting as I dived into Microsofts newest platform; Microsoft Bot Framework. With many companies already creating their own bots using the Bot Framework such as Ebay, Skyscanner, StubHub and Foursquare, I believe that in the next few years every major business will have a chatbot to deal with customers’ basic queries and maybe even go as far as processing full orders. Chatbots have the potential to replace call centres, customers can get their queries answered simply by opening their favourite messaging platform and chatting with a bot.

Introduction

Aims

The purpose of this project is to showcase the power of chatbots and how they can be an alternative to using an application or even a website. The chatbots should be easy to use, respond in a timely fashion and be all round user friendly. The bots should make the users interaction as easy and fast as possible to ensure that the users time is not wasted and that they get what they want without any difficulty or misunderstanding from the bot. The conversation should flow and always keep the user in control of the conversation. Users should come away from their experience with the chatbot and think that it was a fun, easy to use and straightforward interaction that would encourage them to come back without any hesitation.

With messaging platforms being the most used type of application in the world, businesses will be looking to take advantage of this and start to develop their own

chatbots to work along with their social media pages. For example, a person calling a restaurant to see what time they open at or what is the special today, the customer can simply message the page on Facebook and the bot will respond accordingly. This frees up time for real employees to do other work and allows the chatbot to handle the simple tasks. Since users will already have a messaging app installed on their mobile device, there is no need to download a separate application to use the chatbot. This can turn a lot of users away as nowadays there is an plethora of applications available and most users will be fed up of having to download an application that they may only use once or twice.

Technologies

This application will be developed using one of Microsoft's newest frameworks; Microsoft Bot Framework. This framework allows for the bots to be built and deployed using Microsoft Azure. Once the bots are hosted on Azure they can be then hosted on various other platforms such as Facebook Messenger, Skype, Slack and more. For the purpose of this project, I will be mainly using Facebook Messenger as a platform to showcase the bots I've developed as it is easily available with the Messenger application and allows for easy testing.

The bots themselves are developed using C# using the Microsoft Bot Builder packages. Microsofts LUIS is a language understanding and interpretation service that I've used to process the data which the chatbot receives and derive the intent from the users message. LUIS works by taking the users input and scoring it against its database of intents and then performing the function of the highest scoring intent. Users can have multiple ways of expressing a single intent and LUIS is there to figure out what the user wants to do.

Multiple packages and APIs have been used in the development of this application including:

SendGrid – Allows the sending of feedback emails entirely within the chatbot.

Microsoft QnA Maker – Allows me to host a FAQ database that a chatbot can then pull information from and use it to answer the users questions.

Microsoft Computer Vision API – Image analysis tool used to determine what content is in a photo and return it to the user. This can be used to identify images and products.

Facebook API – My bots are available with their own Facebook page. These can be accessed by anyone with a Facebook account. According to my research most users will already have a Messenger account with the app downloaded on their mobile device.

Background Research

This section will cover the background research that I have conducted into chatbots and some of the technologies I've explored before settling on Microsoft Bot Framework

Chatbots

A chatbot is an AI agent that can participate in a conversation with a user. Most are equipped with a messenger type interface with an input from a user and an output from the chatbot. The chatbot processes the users input and outputs a reply based on what the user has just sent. It could be a greeting, conversation topic, or even an image.

Most basic chatbots work by matching an users input with a predefined set of dialog. For example, a user saying "Thank you" will result in the chatbot saying "You're Welcome". The predefined set of dialogs can be set up to imitate a normal conversation between two people. Problems can arise when a user says something the chatbot does not recognize, an example could be the user meaning to say "Thank you", but instead says "Thanks a lot", this can confuse the chatbot as it will be looking to match the "Thank you" input with "Welcome". This leads to a lot of manual work by trying to define every combination of a user saying "Thanks".

Modern chatbots are more complex and feature natural language processing that can learn from user inputs. They can access APIs to get information users such as news,

weather, time etc. They can even process orders and make bookings entirely through a chatbot interface.

Chatbots are well suited for mobile devices as messaging is at the heart of a mobile phone. Messaging has come a long way since SMS messages became popularized in the 2000s and is now on the decline. From the years 2011-2015 the usage of SMS in Ireland has dropped by 44%. 3 billion texts in 2011 compared to 1.7 billion texts in 2015. [John Hargan, killbiller.com 2015] Although SMS is experiencing a decline, this doesn't mean that people aren't sending messages anymore, it just means they are using different services. It is fast becoming the norm where a chatbot is easier to use than an application, and businesses are taking advantage of this.

With automation looking to takeover manual labour and factory type jobs, chatbots are starting to make their way into the customer service sector. Call centre and customer service jobs whereby a human worker will work off a script and a set of answers to generic customer queries will soon be replaced by chatbots. Chatbots can be trained and equipped to deal with the everyday needs of a customer, and they can do so at a very little cost. It is also worth mentioning that chatbots can run 24/7 365, giving customers what they need even during the Christmas and public holidays. With the cost of the development of a chatbot ranging anywhere from $3,000 to $10,000, it would be a no-brainer for a company to implement chatbot services to their customer service department. [Oswalt, 2017] Inevitably, there will be situations where a chatbot will not suffice and a customer will have to be redirected to a human representative, but it is still a step forward in cutting down costs and automation.

Another aspect of chatbots that should not be overlooked is the data they can collect. Chatbots are just another stream of data that companies can exploit benefit from. Chatbot conversations can provide everyday user scenarios that can be used for training material for human workers. Chatbots can be used to complete data sets by acquiring information from users. The conversations can be used a way to learn more about the user and build up an advertising profile that can then be used to for targeted advertising and promotions. From large multi-national organizations to the restaurant down the

road, the data a chatbot can collect can be helpful in identifying who your customers are and what they want.

The infrastructure is there for chatbots to thrive as more and more people are using messaging apps every day. There is a wealth of APIs and platforms for chatbots to explore and make use of and bring interesting features and services to users.

AIML

Alice, short for Artificial Linguistic Internet Computer Entity, is a chatbot developed by Richard Wallace using AIML (Artificial Intelligence Markup Language).

When I was first researching chatbots and how to develop them I came across AIML (Artificial Intelligence Markup Language). It's an XML schema used to create intelligent chatbots. One of the world's most known chatbots; A.L.I.C.E, short for Artificial Linguistic Internet Computer Entity, was developing using AIML. At first it seemed to be the best framework to develop my project on. There was lots of material online for me to read and get to understand how everything worked. I started work on my project using this language and made some progress but regrettably it just wasn't going to work out. It was an old language initially released in 2001 and wasn't very flexible. Basic conversations were easy to get up and running but accessing APIs and getting it hosted on social media platforms caused great difficulty. It's archaic architecture didn't stand up to todays modern platforms. I scrapped all of my work on AIML and moved to Microsoft Bot Framework which I stuck with for the remainder of the projects development. In hindsight, I should've moved on earlier.

Microsoft Bot Framework

In January I came across Microsoft Bot Framework and it looked like the perfect platform for my projects development. It supports development in C# and Node.Js. I have worked with C# on many college projects over the years and Visual Studio would be my preferred IDE, so this was a massive factor in deciding what framework I was going to work with. It allows you to connect your chatbot to multiple platforms at ease

as well as hosting on Azure. This allowed me to work on the project in college and upload my code to Azure via GitHub, and then when I get home I could continue my work seamlessly. Bot Framework is one of Microsofts newest projects which means that the documentation and online material is not of the highest standard but it has been exciting to develop on this platform as it grows. Bot Framework is still in its preview phase but since starting working with Bot Framework many more features have been added and Microsoft seem to be really pushing chatbots into the commercial space so I expect to see big things from them in the near future.

Facebook Messenger

Facebook Messenger is the most used messaging application in the world. With 1.2 billion users, it surpasses WhatsApp to be the most used messaging platform in the world. This is one of the reasons why I chose Messenger as my main social media platform to develop my bots on, another reason being that WhatsApp doesn't have an official API for chatbots to take advantage of. Since Facebook owns WhatsApp it may not be long before we see this introduced.

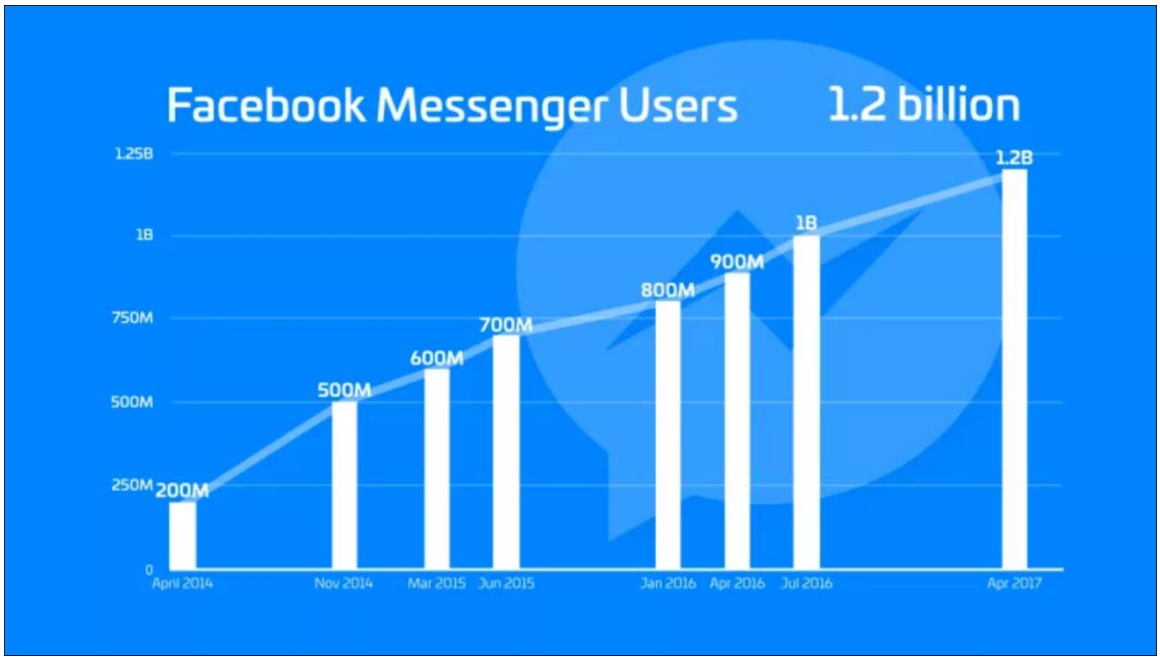


FIGURE 1 MESSENGER USERS [TECH CRUNCH, 2017]

API.AI

Before I decided to work with Bot Framework, I also explored what other frameworks were out there. One of them was API.AI, which was recently acquired by Google. API.AI boasts a wide range of features with support for chatbots, wearables, apps, smart homes and even smart TVs. I started work using the framework but I soon discovered that a lot of the features available with it such as knowledge bases and API packages, were locked behind a paywall. I felt that it was more geared towards a development team and not really catering for a solo developer like myself. As I was just learning about chatbots and looking at my options I chose to stay away from API.AI mainly due to the premium only features. I also found that the documentation was not as easy to read as Microsofts Bot Framework.

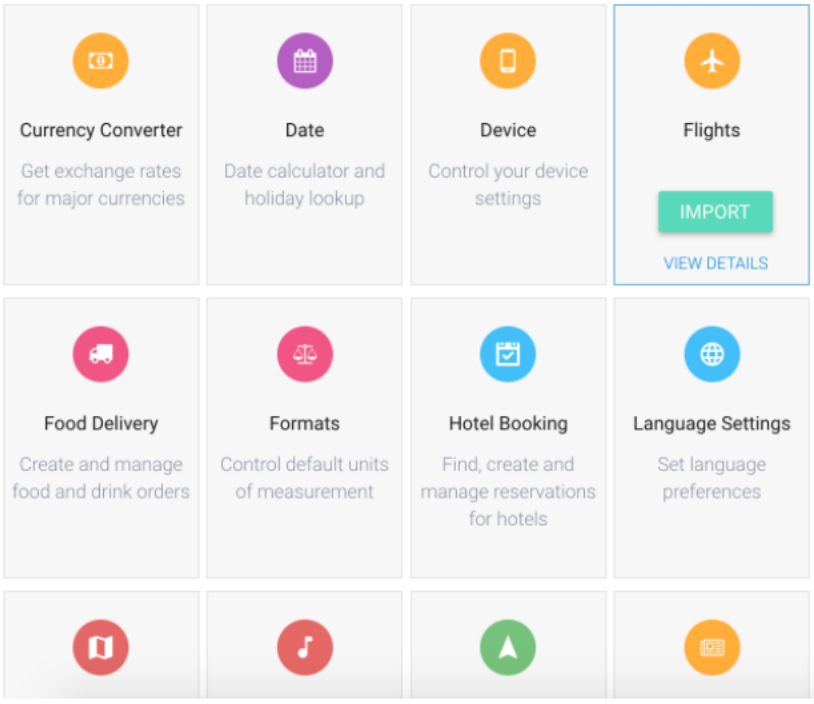


FIGURE 2 API.AI FEATURES

Figure 2 above shows some of the features of their framework. They offer prebuilt bots that you can configure to your needs which would be useful but they are in fact locked

behind a paywall. I also felt that it would kind of take away the fun of working on a project as everything is already defined for you with this framework.

Wit.AI

Like API.AI, Wit.AI is another framework I explored before deciding to use Bot Framework for my project. Wit.AI was acquired by Facebook, you can start to see the trend with all the major companies getting on board with Artificial Intelligence and language processing. Wit.AI works similar to the way Microsofts LUIS works by extracting the intent from a user's message. This seems to be the direction most natural language processing entities are heading. With users seeking to use a chatbot solely for function instead of a conversation, deriving the intents of the user is the chatbots utmost priority. As well as API.AI, Wit.AI features support for bots, apps, smart homes, wearable devices and even robots.

Wit.AI mainly focuses on users building their bots within their web client, a feature which I was trying to stay away from as I would prefer to actually write the code needed. As well as the web client, they offer documentation for development using Node.Js, Python and Ruby. All of which I am not too familiar with so this pushed me away from this framework and left me with Bot Framework. Bot Framework also supports Node.js but it has support for C#, a language I am familiar with and have used to develop many college projects throughout the years.

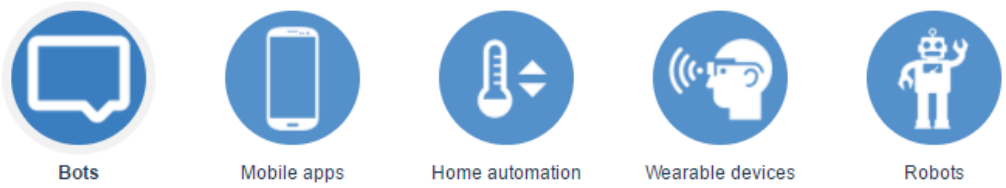


FIGURE 3 FEATURES OF WIT.AI [WIT.AI]

System

Requirements

When setting out to gather my requirements I conducted surveys relating to the use of chatbots on mobile devices or web clients. From these surveys, I was able to establish that the majority of those questioned would be more inclined to use a messaging application on their smart phone as opposed to using it on Desktop/Laptop. I found that in the age range of 12-18 almost all users would only use a messaging application on a mobile device while users in the age range of 19-30 would use both a mobile device as well as a Desktop/Laptop. This may be due to the older using owning a Desktop/Laptop for college/work, while the more younger users rely solely on their mobile device for communication purposes.

With this information, I was able to conclude that the majority of users would prefer a chatbot that would function on a mobile platform. I have structured my requirements based around this information.

Functional requirements

They key functional requirements can be identified as the following:

The chatbot must respond when a conversation is initiated.

This requirement at its most basic level is the core functionality of the bot. The bot must respond to the user when enters the conversation. This sets the conversation up for the user and allows the them to use the bot as they desire. As this is a fundamental requirement for the application to work, it is considered a high priority.

Code Segment

```
else if (message.Type == ActivityTypes.ConversationUpdate)
{
    string replyMessage = string.Empty;
    replyMessage += $"Hello\n\n";
    replyMessage += $"I am a Chat Bot created by Karl Lyons.. \n";
    replyMessage += $"To use the NCI Faq service type faq and type your question... \n";
    replyMessage += $"Type cancel at anytime to return to normal dialog.. \n";

    return message.CreateReply(replyMessage);
}
```

FIGURE 4 REQ 1 CODE SEGMENT

The chatbot must work with Facebook Messenger.

This requirement requires the chatbot to work with Facebook Messenger. This requires the bot to be configured with a Facebook Page, obtaining a PageId, App Id, App secret and a page access token. All of this information needs to be fed into the Bot Framework website to get it configured for access with Facebook Messenger. This requirement gives the bot portability and cross platform functionally by giving it both access to Facebooks Mobile Messenger as well as the web client. From my market research, it was clear the users would like the bot to have a Messenger presence rather than a standalone app.

The chatbot must work on a web client.

This requirement requires the chatbot to work on a web client. From my market research, it was noted that although the majority would prefer a mobile client, there was a number of people who would also like to see a web client for the bots. The bots need to be configured on the Bot Framework platform to allow webchat embedding. This is achieved by hosting the bots on my website located at:

http://karlfyp.azurewebsites.net

The chatbot must work on a mobile device (Facebook Messenger)

This requirement requires the chatbot to function on a mobile device, in this case, using the Facebook Messenger application. It is required to work on both Android and iOS. This allows mobile users to avail of the chatbots services.

The chatbot must work with LUIS.

This requirement requires the chatbot to be configured with a LUIS account subscription. This handles the natural language processing of the bot and the routing of the dialogs. Once a LUIS account is set up and a language model is defined with intents and utterances, the model ID and subscription key can be entered into the RootLUISDialog class. This allows the bot to function using the services of LUIS.

Code Segment

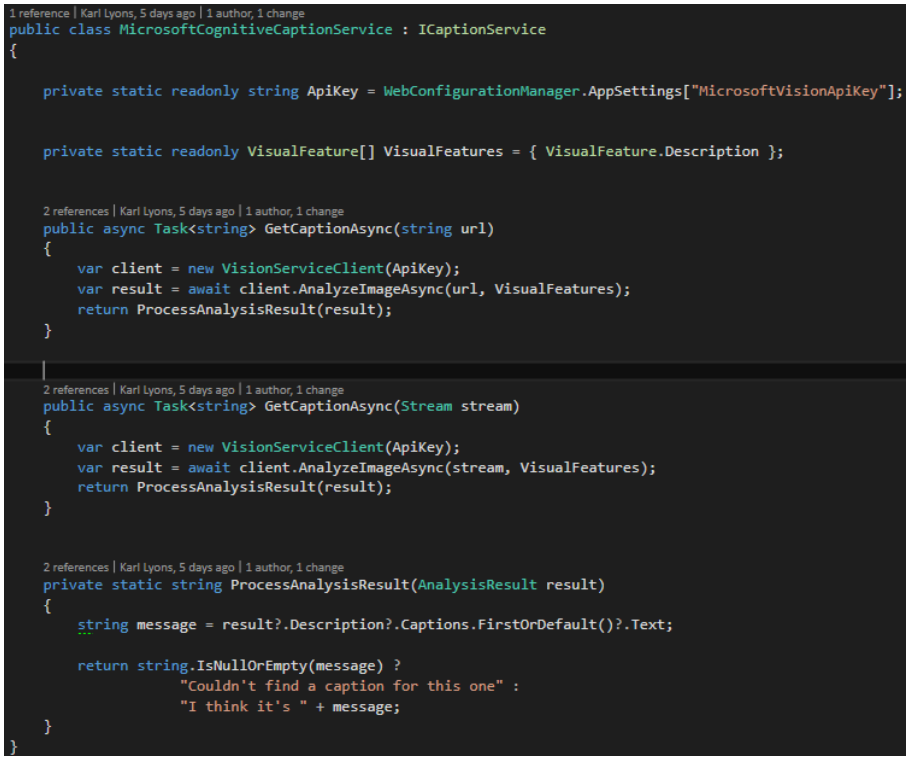
```
[LuisModel("0efe5637-308d-49b8-8d6f-c17c909242cc", "8f885e58d15448eba94240e93225ada4")]
[Serializable]
2 references | Karl Lyons, 2 hours ago | 1 author, 1 change
public class RootLuisDialog : LuisDialog<object>
{

    1 reference | Karl Lyons, 2 hours ago | 1 author, 1 change
    public RootLuisDialog(params ILuisService[] services) : base(services)
    {
    }


    [LuisIntent("")]
    0 references | Karl Lyons, 2 hours ago | 1 author, 1 change
    public async Task None(IDialogContext context, LuisResult result)
    {
        var dialog = new GeneralResponseDialog();
        dialog.InitialMessage = result.Query;
        context.Call(dialog, AfterGeneralResponseHandled);
    }
```

FIGURE 5 REQ 5 CODE SEGMENT

The chatbot must work with Computer Vision API

This requires the chatbot to have a valid Computer Vision API key. This can be obtained from Microsoft Azure Service, once acquired it can be added to the Web.config file so it can be referenced by other classes. This allows the chatbot to access the Computer Vision API and make use of its services.

Code Segment



```
1 reference | Karl Lyons, 5 days ago | 1 author, 1 change
public class MicrosoftCognitiveCaptionService : ICaptionService
{

    private static readonly string ApiKey = WebConfigurationManager.AppSettings["MicrosoftVisionApiKey"];


    private static readonly VisualFeature[] VisualFeatures = { VisualFeature.Description };


    2 references | Karl Lyons, 5 days ago | 1 author, 1 change
    public async Task<string> GetCaptionAsync(string url)
    {
        var client = new VisionServiceClient(ApiKey);
        var result = await client.AnalyzeImageAsync(url, VisualFeatures);
        return ProcessAnalysisResult(result);
    }

    2 references | Karl Lyons, 5 days ago | 1 author, 1 change
    public async Task<string> GetCaptionAsync(Stream stream)
    {
        var client = new VisionServiceClient(ApiKey);
        var result = await client.AnalyzeImageAsync(stream, VisualFeatures);
        return ProcessAnalysisResult(result);
    }


    2 references | Karl Lyons, 5 days ago | 1 author, 1 change
    private static string ProcessAnalysisResult(AnalysisResult result)
    {
        string message = result?.Description?.Captions.FirstOrDefault()?.Text;

        return string.IsNullOrEmpty(message) ?
                    "Couldn't find a caption for this one" :
                    "I think it's " + message;
    }
}
```

FIGURE 6 REQ 6 CODE SEGMENT

The chatbot must work with SendGrid Email API

This requirement allows the chatbot to make use of SendGrid email API to let the users give feedback within the chatbot. The bot must have a valid API key for SendGrid. Once acquired it can be added to the Web.config file where it can then be referenced by other classes. This requirement allows users to give feedback and also give their email to receive a response to their feedback.

Code Segment

```
public class FeedbackForm
{

    [Prompt(new string[] { "What is your name?" })]
    3 references | Karl Lyons, 12 days ago | 1 author, 1 change
    public string Name { get; set; }

    [Prompt("Can I have your email address to reply to you?")]
    3 references | Karl Lyons, 12 days ago | 1 author, 1 change
    public string Contact { get; set; }

    [Prompt("What is your feedback?")]
    2 references | Karl Lyons, 12 days ago | 1 author, 1 change
    public string Feedback { get; set; }

    1 reference | Karl Lyons, 12 days ago | 1 author, 1 change
    public static IForm<FeedbackForm> BuildForm()
    {
        return new FormBuilder<FeedbackForm>()
            .Field(nameof(Contact), validate: ValidateContactInformation)
            .Field(nameof(Feedback), active: FeedbackEnabled)
            .AddRemainingFields()
            .Build();
    }
```

FIGURE 7 REQ 7 CODE SEGMENT

The chatbot must work with QnA Maker

This requires the chatbot to have a valid QnA Maker API key. A QnA service needs to be created and once it is established the knowledge base key and subscription key can be placed into the Web.config file where it can be referenced by the classes that need it. This requirement allows the bot to access the external QnA's created on the service.

Code Segment

```
[Serializable]//
[QnAMakerService("88355712475a4f90bfe15a0c331828d2", "79e4c66e-2fc1-4df9-80b1-bf9632adee9e")]
1 reference | Karl Lyons, 2 hours ago | 1 author, 4 changes
public class QnADialog : QnAMakerDialog<bool>
{
    1 reference | Karl Lyons, 2 hours ago | 1 author, 2 changes
    public override async Task NoMatchHandler(IDialogContext context, string originalQueryText)
    {
        context.Wait(MessageReceived);
        await context.PostAsync($"Sorry, I couldn't find an answer for '{originalQueryText}'.");
        context.Wait(MessageReceived);
    }
```

FIGURE 8 REQ 8 CODE SEGMENT

Usability requirements

The main usability requirement is that the chatbot and website are user friendly. The chatbot, in its own nature is user friendly since it is a simple chat interface. Everyone knows how a chatbot works as it is simply like a conversation with their friends. The messages come in on the main screen and there is a text box at the bottom to type in and a send button.

Users must feel comfortable in the conversation and feel in control. To guide the users through the conversations, help messages should be in place to aid the user if they so happen to get stuck or are unsure of what to do. An example of this can be seen with the Pizza bot. If a user types “help”, the following message will be displayed:

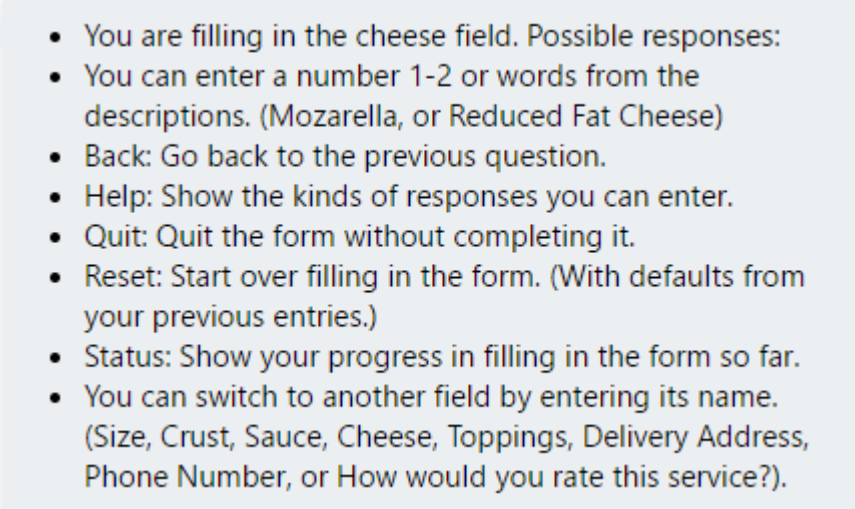


FIGURE 9 PIZZA BOT HELP MESSAGE

This allows the user to view information about the option they are currently at and how they can answer. It tells the user how they can go back, quit, view the status or switch to another option. This more than meets the requirement of letting the user be in control of the conversation. Inexperienced or low skilled users will easily be able to navigate the Pizza bot with the help of the help menu. To ensure that users know about this feature,

I've added it to the welcome message shown below.

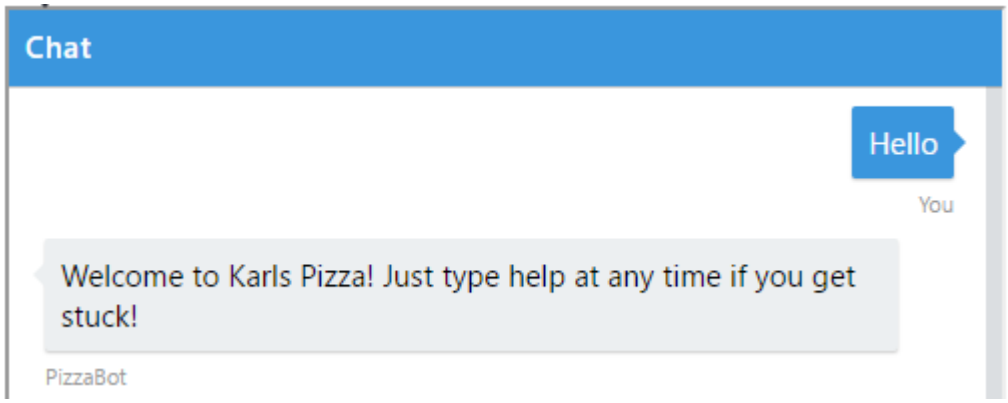


FIGURE 10 PIZZA BOT WELCOME

As with the Pizza Bot, the NCI bot also has some usability features. Users will be greeted with a welcome message as and also how to cancel the current dialog if they get stuck.

The website needs to be user friendly in that any kind of user will be able to open it up and easily be able to navigate through the different chatbots. The website must be accessible to mobile users as well as desktop users. The site has been designed in such a way that it accommodates to both groups of users. To ensure the ease of access of the different chatbots, I have featured them across the navigation bar at the top of the web page. This is shown below.

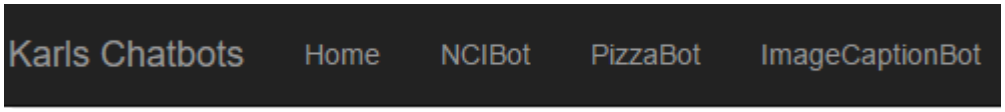


FIGURE 11 WEBSITE NAVIGATIONAL BAR

Facebook Messengers UI is user friendly and easy to navigate. Users that use the chatbots on Messenger will already have a Facebook account and will be familiar with how it works.

Environmental Requirements

The chatbot must be able to run on multiple platforms and operating systems. With the bots hosted on Facebook the bot will be available to anyone with the Messenger application running on their mobile device. Anyone with a computer can access the bots via my website which support all major web browsers (Chrome, Edge, Firefox, Safari, Opera) and runs with no issues. A mobile device can run the chatbots via a mobile web browser but I've found that it does not run as smoothly as in an application or on a computer. I would recommend that users use the application in either Facebook Messenger application or on my website with a computer to get the best experience. When running the website on a mobile browser however, it can be a bit slow to respond compared to other forms of use. Although it is a small bit slower it can still function.

I would recommend that users use the most up to data firmware for their mobile devices and the latest Messenger application.

Non-Functional Requirements

Performance/Response time requirement

Response Times: The Chatbot should reply to the user in a timely manner to keep the conversation flowing without interruption. Users value their time greatly and if the response is not quick enough the user will move on to a different website/app. Google conducted an experiment in regard to load times. They wanted to increase the number of search results shown per page to give the user more information to choose from. They upped the number of search results per page to 30, increasing the load time of the page by 0.5 seconds. Google concluded that traffic dropped by 20% just from a wait

time increase of 0.5 seconds. Users value their time greatly so it is important that the chatbot responds quickly.

Availability requirement

The chatbot must be available to chat at any time. This is achieved with Azure Cloud platform services. All code for the chatbots is uploaded to Azure which is then passed on to whichever messaging platform is in use e.g Facebook Messenger, Skype.

Security requirement

The code must be stored securely by Azure Services and also users information must be kept secure. Facebook and Azure use HTTPS (Secure Socket Layer) for all of their communications. Azure, developed by Microsoft, who has decades of experience and has build some of the most used and powerful software in the world take security very seriously. Microsoft complies with international and industry-specific compliance standards to ensure the security of data.

Reliability requirement

The Chatbot must be consistent with its responses. It must behave consistently in a user-acceptable manner when communicating with a user. The information relayed to the user must be consistent with the dialog topic.

Maintainability requirement

The chatbots must be able to be updated with up to date FAQ's and timetables/exam timetables. This allows the chatbot to develop as time goes on and

Portability requirement

The application should be able to run on any device connected to the internet. The chatbot can be hosted on a variety of social media platforms as well as on personal websites. Once the chatbot is hosted on azure it can then be accessed by other sites and platforms.

Reusability requirement

The AIML files can be used for different Chabot's and can be interchangeable between different bots.

Design and Architecture

This section outlines the design and architecture of the chatbots. The bots are developed using Microsoft Visual Studio 2015. The bots are constructed using Microsoft bot template. The main components of the application are the MessagesController.cs, Web.config, and the Dialog files. The dialog files will vary from bot to bot but they all provide a similar function; to support the dialogs of the application.

Architecture Overview

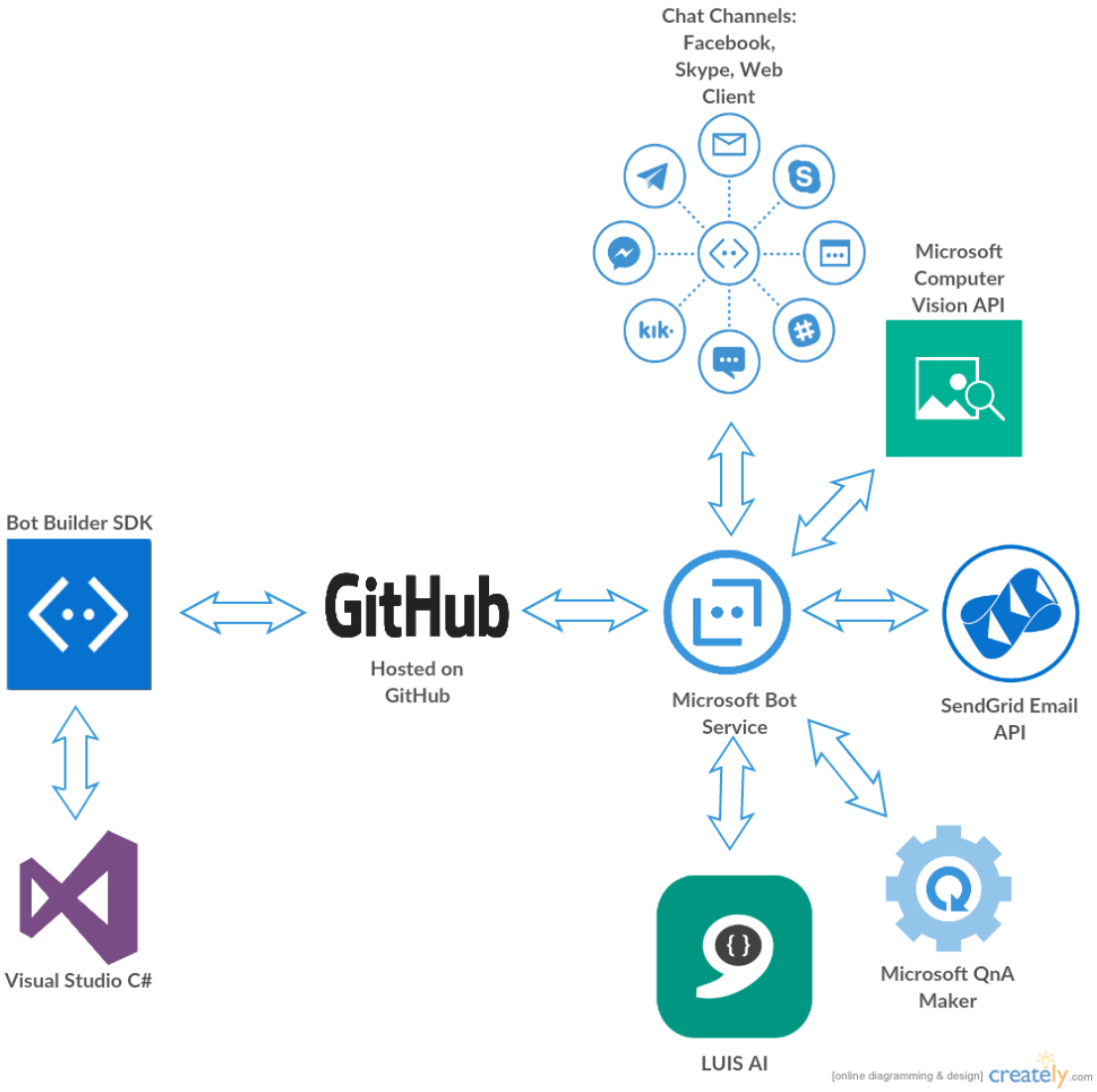


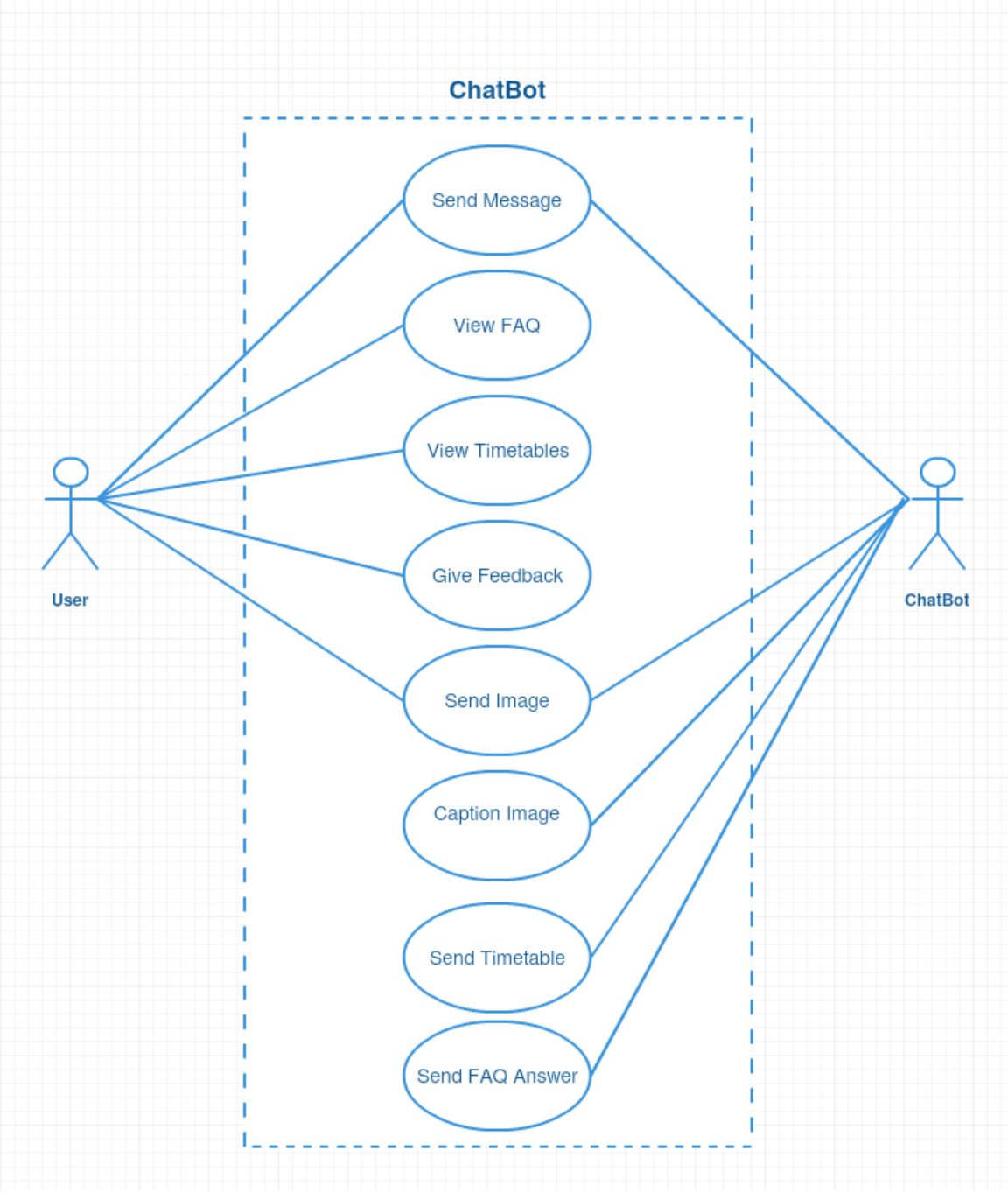
FIGURE 12 ARCHITECTURE OVERVIEW

Shown above is a diagram of the architecture overview of a chatbot. The bots are developed in C# using Visual Studio 2015 with the Bot Builder Source Development Kit.

From there it is pushed up to GitHub and processed my Microsoft Bot Service. This is where it interacts with the various APIs, services and chat channels used in the project.

Use Case Diagram

This diagram demonstrates the typical use of a bot. It shows what the user can do and what the chatbot does to respond.



Initial Setup

In this section I will explain the process of getting a bot hosted online and ready for user interaction.

Once a bot has been created and all of the code is ready to be deployed, it is then pushed to an Azure Web App which hosts the application online. Once it is hosted I can then register the bot with Bot Framework and obtain a Microsoft App Id & Password. Once the bot is successfully registered it can then be set up on the different communication channels. Depending on the purpose of the bot additional setup may be required. For example, with LUIS, an account has to be set up as well as the intents and utterances trained and published. Once the LUIS subscription key is obtained it can be placed in the Web.config file and added to a dialog class.

Application Files

MessagesController

This class is essentially the main class of the application. It deals with the initial set up of the conversation and routes it to the different dialogs associated with the application. For the majority of the bots I've created, the MessagesController class is essential a landing pad for the bot before it is routed off to another dialog. I found it easier to split up the dialogs into a different folder and have separate classes for each different one.

Web.config

The Web.config file holds information such as the BotId, MicrosoftAppID & Password, and other API keys that are in use in this application. Storing them here allows them to be referenced in any other class within the project. Shown below is a snipped of a Web.config file.

```
<configuration>
  <appSettings>
    <!-- update these with your BotId, Microsoft App Id and your Microsoft App Password-->
    <add key="BotId" value="KarlLUISBot" />
    <add key="MicrosoftAppId" value="b000000a-54b8-41b7-a231-9e762c520f2a" />
    <add key="MicrosoftAppPassword" value="QNWek2co4hBv6o8ZpBirqOX" />
    <add key="RecipientEmail" value="karl.lyons@student.ncirl.ie" />
    <add key="SenderEmail" value="7.lyonsy@gmail.com" />
    <add key="SendGridKey" value="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" />
  </appSettings>
```

FIGURE 13 WEB CONFIG FILE

Packages and Main Methods

LUIS is used for natural language processing. It is a language understanding service developed by Microsoft that works be finding the intent of the user. I have incorporated LUIS into the majority of my chatbots as it allows me to figure out what the user intends to do and point them towards the right dialog/method. It allows users to express their intent in a multitude of different ways and still be understood by the chatbot. I feel that this is what makes the chatbots intelligent, they are able to figure out what the user wants no matter how the user phrases it. Below is a sample of a LUIS dialog, in this case, the RootLUISDialog class.

What's happening here with [LUISIntent("Timetable")], means that when LUIS detects that a users intent is to view a timetable, it will execute the StartTimetableDialog method, which then initiates the TimetableDialog class which holds the information and images of the timetables. It also has handler so if they bot doesn't understand the users

input, it will send “Sorry I didn’t understand.”

```
[LuisIntent("Timetable")]
0 references | Karl Lyons, 4 days ago | 1 author, 1 change
public async Task StartTimetableDialog(IDialogContext context, IAwaitable<IMessageActivity> message, LuisResult result)
{
    var faqDialog = new TimetableDialog();
    var messageToForward = await message;
    await context.Forward(faqDialog, AfterTimetableDialog, messageToForward, CancellationToken.None);

}

1 reference | Karl Lyons, 4 days ago | 1 author, 1 change
private async Task AfterTimetableDialog(IDialogContext context, IAwaitable<bool> result)
{
    var messageHandled = await result;
    if (!messageHandled)
    {
        await context.PostAsync("Sorry I didn't understand.");
    }
    context.Wait(MessageReceived);

}

[LuisIntent("Faq")]
0 references | Karl Lyons, 5 days ago | 1 author, 1 change
public async Task StartFaqDialog(IDialogContext context, IAwaitable<IMessageActivity> message, LuisResult result)
{
    var faqDialog = new QnADialog();
    var messageToForward = await message;
    await context.Forward(faqDialog, AfterFaqDialog, messageToForward, CancellationToken.None);

}

1 reference | Karl Lyons, 5 days ago | 1 author, 1 change
private async Task AfterFaqDialog(IDialogContext context, IAwaitable<bool> result)
{
    var messageHandled = await result;
    if (!messageHandled)
    {
        await context.PostAsync("Sorry I didn't understand.");
    }
    context.Wait(MessageReceived);

}
```

FIGURE 14 ROOTLUISDIALOG CLASS

Also in Figure 14, it shows the LUIS intent for Faq. This works the same as the timetable intent, where it initiates the QnADialog class that I will explain later on in this report.

General Response Dialog

This class is used to handle all of the generic parts of a conversation, from hello/goodbye to thankyou/you’re welcome. Below I have shown a sample of the class. It works using the BestMatchDialog which function by having an array of user messages and a reply based on what the user has sent. In the first method seen below, you can see a variety of ways the user can say “Hello”, all of these messages will still get the same response; “Hi there”. This prevents the bot from getting confused if the user means to say hello but phrases it differently. The threshold parameter gives some leeway for spelling mistakes and allows for some slight difference in the phrasing.

```
[Serializable]
1 reference | Karl Lyons, 5 days ago | 1 author, 2 changes
public class GeneralResponseDialog : BestMatchDialog<bool>
{

    [BestMatch(new string[] { "Hi", "Hi There", "Hello there", "Hey", "Hello",
        "Hey there", "Greetings", "Good morning", "Good afternoon", "Good evening", "Good day" },
        threshold: 0.5, ignoreCase: true, ignoreNonAlphaNumericCharacters: false)]
    0 references | Karl Lyons, 6 days ago | 1 author, 1 change
    public async Task HandleGreeting(IDialogContext context, string messageText)
    {
                                          interface Microsoft.Bot.Builder.Dialogs.IDialogContext
                                          The context for the execution of a dialog's conversational process.
        await context.PostAsync("Hi there.");
        context.Wait(MessageReceived);
    }

    [BestMatch(new string[] { "Bye", "cya", "Bye Bye", "See you later", "Goodbye"},
        threshold: 0.5, ignoreCase: true, ignoreNonAlphaNumericCharacters: false)]
    0 references | Karl Lyons, 6 days ago | 1 author, 1 change
    public async Task HandleGoodbye(IDialogContext context, string messageText)
    {
        await context.PostAsync("Bye talk to you soon.");
        context.Wait(MessageReceived);
    }


    [BestMatch(new string[] { "Thank you", "Thanks", "Thanks so much", "Ty",  },
       threshold: 0.5, ignoreCase: true, ignoreNonAlphaNumericCharacters: false)]
    0 references | Karl Lyons, 6 days ago | 1 author, 1 change
    public async Task HandleThanks(IDialogContext context, string messageText)
    {
        await context.PostAsync("You're welcome.");
        context.Wait(MessageReceived);
    }
```

FIGURE 15 GENERAL RESPONSE DIALOG

Global Message Handling

Global message handling allows me to reset the bot and return it to its initial startup state. This is useful when the bot is using multiple dialogs and it can be hard to navigate through them back to the start. By setting up global messaging handling I was able to assign a key word, in this case “Cancel” that returns the bot to its startup dialog. This works by setting the bot to check every message for the keyword “Cancel” before processing the methods that are in place in the current dialog. In the pizza ordering bot, this is very useful as if the user makes a mistake while submitting their order they can simply type cancel to reset their order and continue from the start.

In order to do this, I needed to set up a Cancel class, which looks at user’s messages for the keyword “Cancel”, and resets the dialog. As with the GeneralResponseDialog and QnA dialog, there is a Score method that allows you to set a threshold for the keyword. In this case I do not want the bot to be making a guess at what the user wanted to do so

I've set it to 1.0 to avoid any confusion. This means the dialog will only reset if the user types the key word correctly. I also needed to set up a BotModule class to register the scorable method. Once that is done it also needs to be initialized in the Global.asax class.

QnA Maker is another project of Microsoft used to create FAQs that can link up to bots developed using Bot Framework. I used this to host NCI's FAQ and relay it to my NCI bot. This cuts out adding in each question manually and allows for easy access to update a question or edit an answer simply by accessing the website. This means I can update questions without touching the application code. Below is a screenshot of the code implementing the QnA dialog. An interesting part of this snipped is the LowScoreHandler method. This works by assigning a score threshold, in this case I've set it to 85 as I feel it works the best and leave a bit of leeway in the users phrasing of the question. QnA Maker works similar to LUIS, where it will score the message based on the intent of it and go with the highest scoring one. With QnA maker, if the users question is at least a 85% match to one of the questions in the database, it will then send the answer back to the user. This gives the user some flexibility in the phrasing of the questions. An example could be, "How do I get to NCI"? The bot will respond with the location and transport information. That is how the question is phrased on the website, but if type "Where is NCI?" It will give me the same answer even though the question was phrased differently.

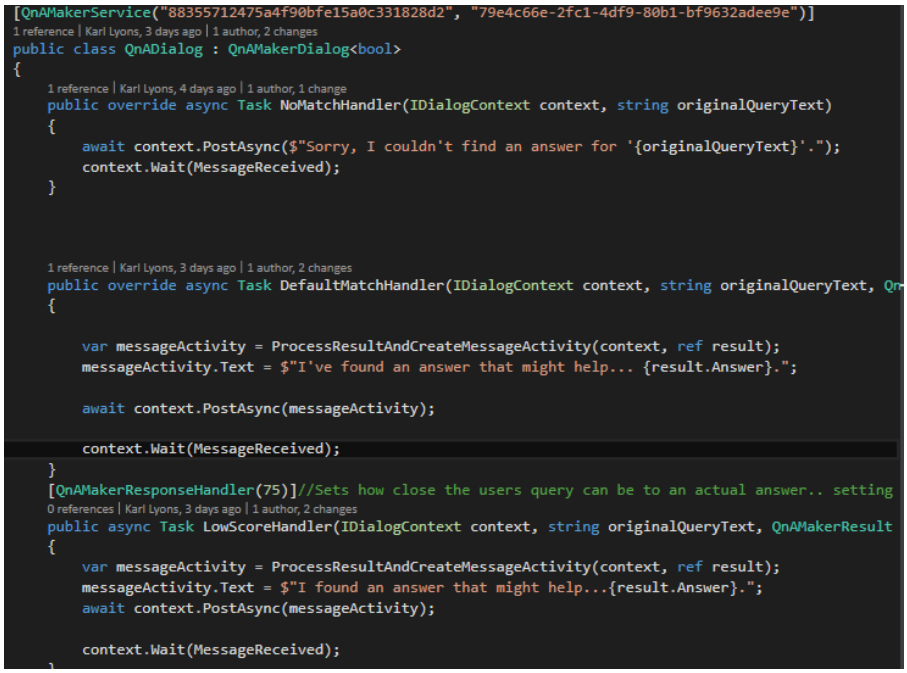


FIGURE 16 QNA CODE

Security

As a cyber security student, I felt that it was fitting that I touch on the security aspects of this project. Unfortunately, there is not very much to cover as since the bots are hosted online with 3rd parties, the security side of things is taken away from me. One feature I did implement was on the chatbot website I created. I added a “RequireHttps” attribute to the Home controller which handles all the bots hosted on the site. This ensures that users have a secure connection when chatting with the bots on the site and can prevent the conversation data from being view or even intercepted.

Although I couldn’t implement many security features to this project I did consider the privacy policy of Facebooks Messenger. Users wishing to use chatbots on Messenger will have to be aware that their conversations data may be being used for research purposes or even to generate an advertising profile of the user. For example with the Pizza bot chatbot, Facebook will store information such as delivery address and payment information. Shown below is an extract from their privacy policy page.

Information about payments.
If you use our Services for purchases or financial transactions (like when you buy something on Facebook, make a purchase in a game, or make a donation), we collect information about the purchase or transaction. This includes your payment information, such as your credit or debit card number and other card information, and other account and authentication information, as well as billing, shipping and contact details.

Device information.
We collect information from or about the computers, phones, or other devices where you install or access our Services, depending on the permissions you've granted. We may associate the information we collect from your different devices, which helps us provide consistent Services across your devices. Here are some examples of the device information we collect:

- Attributes such as the operating system, hardware version, device settings, file and software names and types, battery and signal strength, and device identifiers.
- Device locations, including specific geographic locations, such as through GPS, Bluetooth, or WiFi signals.
- Connection information such as the name of your mobile operator or ISP, browser type, language and time zone, mobile phone number and IP address.

FIGURE 17 FACEBOOK PRIVACY POLICY

Also shown in this extract is that Facebook tracks device information and even your location. It is important that users understand that they are handing their data over to Facebook when they make use of the chatbots hosted on their site.

Implementation

Implementation was the hardest part of this project. I was working with a Framework that I never used before and was still in development. It took me a lot of trial and error and a lot of reading to really understand Microsofts Bot Framework. Once I had a bot set up and ready to be worked on one of the hardest things I found was trying to control the dialogs, meaning that when I started to further develop the bots to be multi-functional, I ran into difficulties with that. Working on a bot that one main function was fairly easy once I knew what I had to do. As soon as more functionality was added some issues started to arise. Some of the issues included bots getting stuck on the same dialog, or sometimes even just getting into a loop of saying “Sorry I couldn’t understand you.”

An example of this issue is with the NCI bot. I could implement the FAQ part fine and that caused no issues but the bot would get confused when the user was looking for something other than the FAQ, even something as simple as saying “Hello” or “Goodbye” could sometimes cause the bot to get stuck and crash.

To overcome this, in the Messages Controller(Where the conversation starts off, landing pad for the user and bots conversation), as soon as a conversation is initiated I send the welcome message (Figure 16),

```
else if (message.Type == ActivityTypes.ConversationUpdate)
{
    string replyMessage = string.Empty;
    replyMessage += $"Hello\n\n";
    replyMessage += $"I am a Chat Bot created by Karl Lyons.. \n";
    replyMessage += $"To use the NCI Faq service type faq and type your question... \n";
    replyMessage += $"Type cancel at anytime to return to normal dialog.. \n";

    return message.CreateReply(replyMessage);
```

FIGURE 18 WELCOME MESSAGE

The conversation then heads towards the RootLUISDialog class. (Figure 18)

```
public async Task<HttpResponseMessage> Post([FromBody]Activity activity)
{
    if (activity.Type == ActivityTypes.Message)
    {
        await Conversation.SendAsync(activity, () => new RootLuisDialog()); //Sends straight to Root LUIS dialog
    }
```

FIGURE 19 SEND TO ROOT

Once in the RootDialog class, the conversation could go a number of ways. After the initial greeting the bot is then looking out for a number of intents including: Feedback, Timetable, FaQ and a blank intent. LUIS intent methods are set up to analyse the users next message and extract the intent from it. If the intent matches one of these methods, they are sent to the corresponding dialog where they can continue the conversation. If not, that is what the blank dialog is for. Shown below is the LUIS intent method for “” or blank. This is when LUIS cannot figure out what the user wants or the user is just saying hello or how are you.

```
[LuisIntent("")]
0 references | 0 changes | 0 authors, 0 changes
public async Task None(IDialogContext context, LuisResult result)
{
    var dialog = new GeneralResponseDialog();
    dialog.InitialMessage = result.Query;
    context.Call(dialog, AfterGeneralResponseHandled);
}

1 reference | 0 changes | 0 authors, 0 changes
private async Task AfterGeneralResponseHandled(IDialogContext context, IAwaitable<bool> result)
{
    var messageHandled = await result;

    if (!messageHandled)
    {

        await context.PostAsync("I’m not sure what you want");

    }

    context.Wait(MessageReceived);
}
```

FIGURE 20 LUIS BLANK INTENT

This method then checks the users messages for a list of responses located in the GeneralResponse Dialog. This fixed the issue of having the bot crash after a user would simply say “hello” or “how are you”. A sample of that class is shown below.

```
[BestMatch(new string[] { "Thank you", "Thanks", "Thanks so much", "Ty",  },
   threshold: 0.5, ignoreCase: true, ignoreNonAlphaNumericCharacters: false)]
0 references | Karl Lyons, 11 days ago | 1 author, 1 change
public async Task HandleThanks(IDialogContext context, string messageText)
{
    await context.PostAsync("You're welcome.");
    context.Wait(MessageReceived);
}

[BestMatch(new string[] { "How are you?", "How's things?", "How's you?", "Are you okay?", },
   threshold: 0.5, ignoreCase: true, ignoreNonAlphaNumericCharacters: false)]
0 references | Karl Lyons, 11 days ago | 1 author, 1 change
public async Task HandleHowareyou(IDialogContext context, string messageText)
{
    await context.PostAsync("I'm great what about you?");
    context.Wait(MessageReceived);
}

[BestMatch(new string[] { "I'm good", "I'm good thanks", "Good thank you", "Good thanks", "I'm great",
    "I'm not good", "im good thanks"
                                                                                    },
   threshold: 0.5, ignoreCase: true, ignoreNonAlphaNumericCharacters: false)]
0 references | Karl Lyons, 10 days ago | 1 author, 2 changes
public async Task HandleHowareyouReply(IDialogContext context, string messageText)
{
    await context.PostAsync("That's good to hear.");
    context.Wait(MessageReceived);
}
```

FIGURE 21 GENERAL RESPONSE DIALOG

This class using the BestMatch Dialog package created by Gary Pretty, a Microsoft MVP who focuses on Microsoft Bot Framework. A lot of his tutorials have been a great help in the development of this project as there is a lack of tutorials online up to the standard of Garys. This package allows me to set a number of strings that could match a users message and set a response to be used if there is a match between the strings and the users message. It also allows me to set a parameter for the threshold. This allows there to be some space for error if the users message doesn't fully match up with the string. This prevents the bot from getting confused when the user simply spells a word wrong or has an extra space in the message.

Next step was to implement the QnA dialog. To do this I used Microsofts QnA Maker. First I needed to create a new QnA service and import the questions from NCI's FAQ page. With the service created and trained, it can then be published. Once published I receive the knowledge base ID and subscription key. This is then used in the QnADialog class shown below.

```
namespace KarlLUISBot.Dialogs
{
    [Serializable]//
    [QnAMakerService("88355712475a4f90bfe15a0c331828d2", "79e4c66e-2fc1-4df9-80b1-bf9632adee9e")]
    1 reference | Karl Lyons, 6 days ago | 1 author, 3 changes
    public class QnADialog : QnAMakerDialog<bool>
    {
```

FIGURE 22 KNOWLEDGE & SUBSCRIPTION KEY

Now I needed to add methods to handle the situation whereby the bot cannot find an answer to the users question and also a close match method, which will allow for some differences in the phrasing of the question. The method for no match is shown below.

```
public override async Task NoMatchHandler(IDialogContext context, string originalQueryText)
{
    await context.PostAsync($"Sorry, I couldn't find an answer for '{originalQueryText}'.");
    context.Wait(MessageReceived);
}
```

FIGURE 23 NO MATCH HANDLER

The close match method allows me to set parameter for how close the users message can be to the original question. After much testing, I found that 85 seemed to work best. Allowing for small differences in phrasing but not too much to get confused with other similarly worded questions. This method is shown below.

```
[QnAMakerResponseHandler(85)]//Sets how close the users query can be to an actual answer.. setting at 50 can give some off answers..
0 references | Karl Lyons, 6 days ago | 1 author, 3 changes
public async Task LowScoreHandler(IDialogContext context, string originalQueryText, QnAMakerResult result)
{
    var messageActivity = ProcessResultAndCreateMessageActivity(context, ref result);
    messageActivity.Text = $"I found an answer that might help...{result.Answer}.";
    await context.PostAsync(messageActivity);

    context.Wait(MessageReceived);
}
```

FIGURE 24 CLOSE MATCH METHOD

The next feature to implement was the Timetables Dialog. This feature allows the user to search for their timetable and view an image of it. I tried multiple ways to do this. One of them was to have the images stored within the application and map each image to a query. For example, “Computing Cyber Security” would display the cyber security timetable. I was having difficulty getting the images to load correctly. Another way was

hosting the image online and pointing the bot towards them but again I was having issues with them loading correctly. I finally settled on working with QnA Maker. Although it primarily functions by creating FAQ, it allowed me to set key words and match an answer to it. One issue with this was that QnA maker did not allow images to be hosted within its message, so I needed to host my timetable image online. This was fairly straightforward so I uploaded the timetable images to Imgur and added the answers to the corresponding questions.

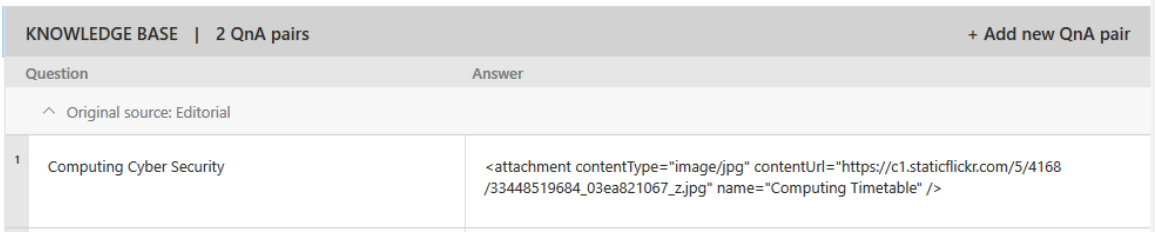


FIGURE 25 TIMETABLE QNA

I started to test the bot and the images we're not showing, the link given led to an image that didn't exist. After much time spent trying to figure out what the issue was, I used Bot Framework Emulator to see what was going on with the messages sent. Using the Emulator I was able to examine each message and see its JSON properties. I noticed that when QnA maker processes image links, it converts the text to all lower case. This causes issues with images hosted on Imgur as the links are case sensitive. So, to overcome this I needed to find an image hosting site that provides its links with all lower-case letters or even just numbers. After trying multiple different hosting sites Flickr was the one that did the job. The image links used contain numbers and a lowercase letter so it is perfect for my needs. With the images loading correctly I was now able to add the rest of the timetables to the service. Shown below is the method that shows the timetable to the user.

```
public override async Task DefaultMatchHandler(IDialogContext context, string originalQueryText,
    QnAMakerResult result)
{
    //keeps the markup
    context.Wait(MessageReceived);
    var messageActivity = ProcessResultAndCreateMessageActivity(context, ref result);
    messageActivity.Text = $"Here is your timetable... /n {result.Answer}.";

    await context.PostAsync(messageActivity);

    context.Wait(MessageReceived);
}
```

FIGURE 26 TIMETABLE METHOD

With the issue of general response dialogs out of the way and the timetables and QnA implemented, I was able to have a bot that was multi-functional and could reply to general messages from the user. The next issue I faced which I foolishly overlooked for quite some time, when a user wants to use the FAQ but then go back and check a timetable the bot was unable to do so and was locked into the FAQ dialog. When I was initially testing the bot I was just checking that I could use the timetable okay and then I would compile and see if I could use the FAQ.

To solve this issue I needed to create a way for the user to return to the root dialog at any time. Unfortunately, with the FAQ dialog, if I just said back the bot would not understand the question and just ask the user to repeat the question. I found a tutorial online, by Gary Pretty that had exactly what I needed. Global Message Handling. This allowed me to get the bot to check every message for a keyword before it was processed by the respective dialog class. In this case I used the word cancel. To implement this, two new classes need to be created to handle this and also a package needed to be imported. It uses the Bot Framework Scoreable package. This package is primarily used to score a user message based on a given parameter. In the case of global message handling, I get it to check the users messages for the word “cancel”, and if it matches up 1:1 it will reset the dialog. A sample from the CancelScorable class is shown below. The second method show resets the dialog.

```
protected override double GetScore(IActivity item, string state)
{
    return 1.0;
}

0 references | Karl Lyons, 10 days ago | 1 author, 1 change
protected override async Task PostAsync(IActivity item, string state, CancellationToken token)
{
    this.task.Reset();
}
0 references | Karl Lyons, 10 days ago | 1 author, 1 change
protected override Task DoneAsync(IActivity item, string state, CancellationToken token)
{
    return Task.CompletedTask;
}
```

FIGURE 27 SCOREABLE

This allows the user to back out of a dialog at any given time simply by entering the keyword. It provides more flexibility to the conversation.

With my Pizza Bot, I needed to implement a form that the users can fill out with their order. I needed the form to be flexible and cater to many several types of orders and to provide a clear and uncomplicated way to for the user to do this. To do this I used FormFlow, one of the features of Microsofts Bot Framework. This feature was fairly easy to implement as I found the documentation on FormFlow to be very clear and provided lots of examples. The first step to implementing this feature was to populate the form with the various options needed for a pizza including size, crust, sauce, type of cheese, toppings etc. Next step was to build the form, add some welcome messages, add the prompts for each choice, a confirmation step towards the end of the dialog and finally some validation for the delivery address and phone number. Shown below is the build form method.

```
public static IForm<PizzaOrder> BuildForm()
{
warning disable 1998//Error with the validation methods for numbers and address
    OnCompletionAsyncDelegate<PizzaOrder> processOrder = async (context, state) =>
    {
        await context.PostAsync("Thank you for choosing Karls Pizza! We are currently processing your order.");
    };
    return new FormBuilder<PizzaOrder>()
            .Message("Welcome to Karls Pizza! /n" +
                     "Just type help at any time if you get stuck!")
                .Field(nameof(Size))
                .Field(nameof(Crust))
                .Field(nameof(Sauce))
                .Field(nameof(Cheese))
                .Field(nameof(Toppings))
            .Field(nameof(PizzaOrder.DeliveryAddress),
            validate: async (state, response) =>
            {
                var result = new ValidateResult { IsValid = true, Value = response };
                var address = (response as string).Trim();
                if (address.Length > 0 && (address[0] < '0' || address[0] > '9'))// prevents dodgy addresses from being inputted
                {
                    result.Feedback = "Address must start with a number.";
                    result.IsValid = false;
                }
                return result;
            })
            .Field(nameof(PhoneNumber),
            validate: async (state, response) =>
            {
                var result = new ValidateResult { IsValid = true, Value = response };
                var number = (response as string).Trim();
                if (number.Length > 10 || (number.Length < 10))
                {
                    result.Feedback = "Your number must be 10 digits long.";// prevents incorrect numbers
                    result.IsValid = false;
                }
                return result;
            })
            .Confirm("Is everything correct?")
            .AddRemainingFields()
            .OnCompletion(processOrder)
            .Build();
}
```

FIGURE 28 PIZZA FORM

Shown in the snippet of the pizza form code is the validation methods for the address and phone number. This prevents users from inputting an invalid address and also an invalid phone number. Phone numbers are required to have 10 digits.

Another chatbot I created was the Image Caption bot. This bot uses Microsofts Computer Vision API, so my first step was to get registered there and grab an API key to use with my bot. Once that was acquired I could place it into the appSettings.config file. From there it can be referenced throughout the application. By following the tutorial on Microsofts website, I was able to correctly implement this feature. Some highlights of the code are shown below.

```
private async Task<Activity> HandleSystemMessage(Activity activity)
{
    switch (activity.Type)
    {
        case ActivityTypes.DeleteUserData:

            break;
        case ActivityTypes.ConversationUpdate:
            // What the user is sent when it enters the conversation
            if (activity.MembersAdded.Any(m => m.Id == activity.Recipient.Id))
            {
                var connector = new ConnectorClient(new Uri(activity.ServiceUrl));

                var response = activity.CreateReply();
                response.Text = "Welcome to Karls Chatbot. This chatbot using Microsofts Vision API to try and figure what is in a picture." +
                                " Send me a picture and I will try and guess what it is.";

                await connector.Conversations.ReplyToActivityAsync(response);
            }
```

FIGURE 29 IMAGE CAPTION WELCOME MESSAGE

Shown above is the welcome message sent to the user when it enters the conversation. Shown below is the process result method which also has handling for a situation whereby the bot cannot find a caption for the image.

```
private static string ProcessAnalysisResult(AnalysisResult result)
{
    string message = result?.Description?.Captions.FirstOrDefault()?.Text;

    return string.IsNullOrEmpty(message) ?
                "Couldn't find a caption for this one" :
                "I think it's " + message;
}
```

FIGURE 30 IMAGE CAPTION PROCESS

Finally, my next task was to get all of my bots hosted on my own website which I could use to demonstrate. This gave me no challenge as I simply set up an MVC websiteusing Visual Studio and got it hosted with Azure. With the website created I just needed to edit the home page to my liking and display a link to each of the chatbots in the navigational bar. One extra feature I did implement into the website was the “RequireHttps” attribute. This ensured that all of the requests to and from this site were completed using HTTPS. If a user did not have a HTTPS connection they would not be able to use the chat bots. I’ve touched on this in the security section of this report.

Testing

Bot Framework Emulator

The bot framework emulator is a desktop application that can be used to test the bots created with Microsoft Bot Framework. This tool allows me to test my bots locally or even remotely. It emulates a normal conversation but also shows the JSON information that is sent and received with each message. The emulator also gives me the opportunity to test my chatbots locally before they are pushed online and in reach of users. I can test the accuracy of the bots replies when asking different questions to see what responses may need to be updated/modified. Having a sandbox to really dig into the bot and see how it performs with a range of different queries has helped greatly in the development of this project. This information gives an insight into how the bot is working and has been vital for the testing of my project.

Figure 31 shows the emulator in action. In this instance, I am testing the responses of the FAQ bot. The panel on the right hand side shows the JSON information received in the messages. Using the emulator I am able to test the accuracy of the FAQ bot. In this example, I've asked "how do I get to NCI?" and "Where is NCI?". Both questions return the same answer. This test reveals that the bot is able to figure out the intent of the

users message regardless of the phrasing.

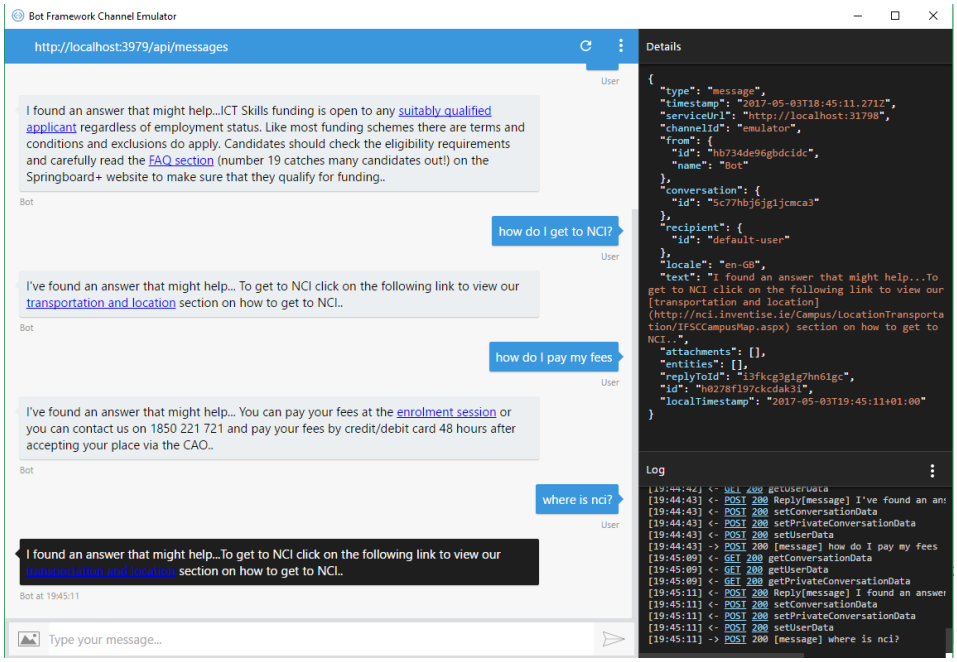


FIGURE 31 EMULATOR TESTING

Test Cases

Due to the unique nature of this project I decided to conduct some conversational tests. I will outline a conversational path I wish to take and then test it against a live bot and see how it plays out.

For testing purposes, I will initiate a conversation with the Pizza bot and go through the steps, but attempt to alter my choice mid order, to see if I can alter my order successfully with no difficulty. In this case I will go through my order as normal and then change the size of the pizza before the order is complete. Shown below is the alteration of the order.

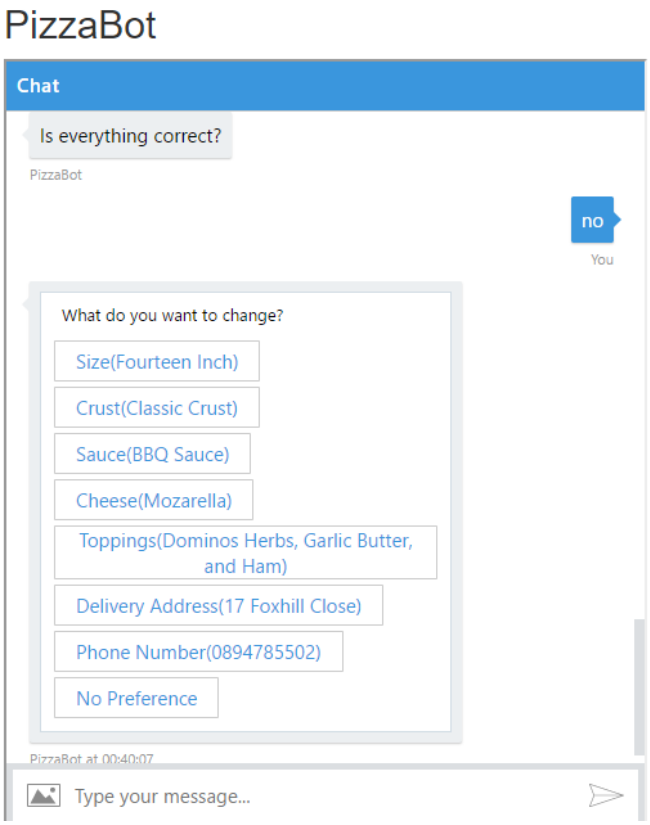


FIGURE 32 ORDER ALTERATION

With the size changed, I could fully complete the order.

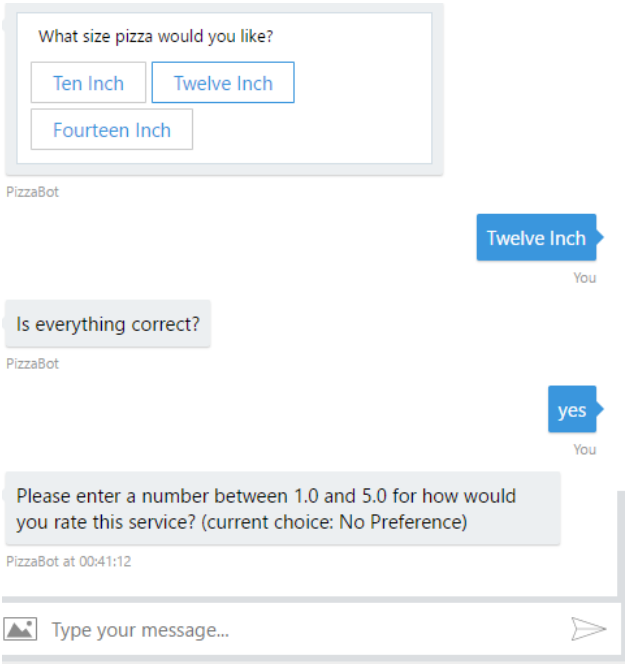


FIGURE 33 UPDATED ORDER

Manual Test Cases

Shown below is the manual test cases conducted for this project. These scenarios cover the main functional requirements of this project. The tests were mainly conducted on my website and also using the Facebook messenger client on both iOS and Android platforms.

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass/Fail
T01	Initiate Conversation with bot. Requirement #1	1. Go to: http://karlfyp.azurewebsites.net/Home/NCIBot 2. Wait for welcome message from bot	Not applicable	Bot will initiate conversation with welcome message.	Bot sends welcome message.	Pass.
T02	Facebook Messenger Test Requirement #2	1. Go to: https://www.facebook.com/ChatBot-1653415881629800/ 2. Click Message to open messenger 3. Wait for welcome message from bot	Facebook Email Password	Bot should send welcome message when chat is opened.	Bot sends welcome message when chat is opened.	Pass.
T03	Web Client Test Requirement #3	1. Go to: http://karlfyp.azurewebsites.net/Home/NCIBot 2. Check that web client is embedded and active.	Not Applicable	Bot should be available to chat with on the web client	Bot was available on the web client.	Pass.
T04	iOS Messenger Test Requiremnt #4	1. Download Messenger Application from App Store 2. Open application and login 3. Message chatbot page 4. Await reply	Facebook Email Password	Bot should reply to user	Bot replied after user messaged.	Pass.
T05	Android Messenger Test Requiremnt #4	1. Download Messenger Application from Google Play Store 2. Open application and login 3. Message chatbot page 4. Await reply	Facebook Email Password	Bot should reply to user	Bot replied after user messaged.	Pass.
T06	Chatbot Test with LUIS Requirement #4	1. Go to: http://karlfyp.azurewebsites.net/Home/NCIBot 2. Initiate Conversation 3. Test for a LUIS intent 4. Type "Faq" to trigger LUIS intent for Qna	Not Applicable	Bot should initiate the Faq Dialog.	Bot initiates the Faq Dialog and is ready for questions.	Pass.
T07	Chatbot Test with Computer Vision API Requirement #5	1. Go to: http://karlfyp.azurewebsites.net/Home/ImageCaptionBot 2. Initiate Conversation 3. Send Image 4. Await reply	Any kind of Image	Bot should reply with information on the image.	Bot replies with their guess of what the image is.	Pass.
T08	Chatbot Test with SendGrid Email API Requirement #6	1. Go to: http://karlfyp.azurewebsites.net/Home/NCIBot 2. Initiate Conversation 3. Type "Feedback" 4. Follow steps given by the bot 5. Send feedback	Email Address Feedback data	After user completes steps on screen the feedback should be sent.	Feedback successfully submitted.	Pass.

FIGURE 34 MANUAL TEST CASE 1

T09	Chatbot Test with QnA Maker	1. Go to:	Question	Bot should reply	Bot replies and	Pass.
		http://karlfyp.azurewebsites.net/Hom		with an answer	answers the question	
		e/NCIBot		relating to the	to the best of its	
		2. Initiate Conversation		question asked.	ability	
		3. Type "Faq" to trigger QnA Dialog				
		4. Type question related to NCI				
		5. Await reply				

FIGURE 35 MANUAL TEST CASE 2

Selenium

I attempted to create automated tests using Selenium Firefox IDE but unfortunately it was not compatible with the web chat client. The tests passed in the selenium IDE which is shown below in figure 36.

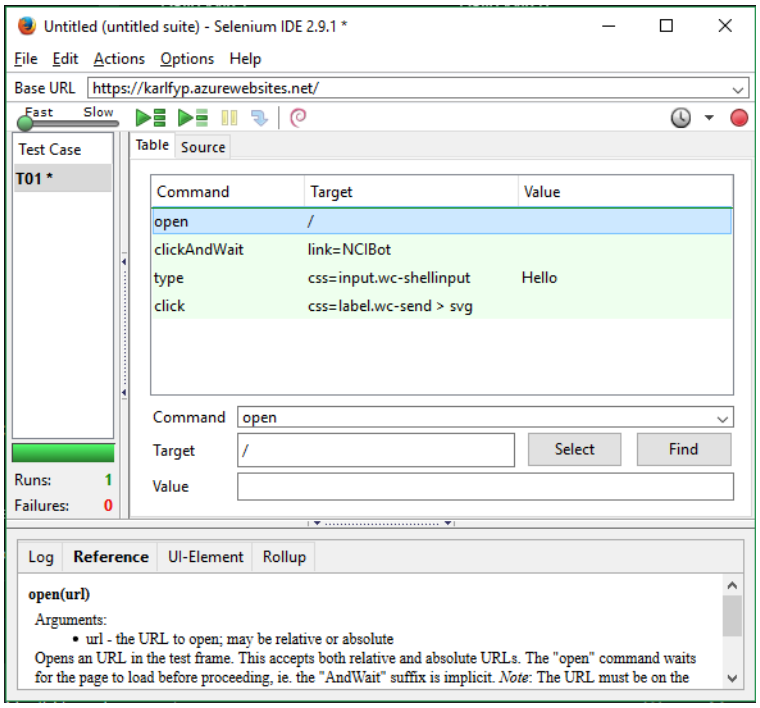


FIGURE 36 SELENIUM IDE TEST

When I extracted the test case to Visual Studio, I configured the class by adding in sleep commands "`Thread.Sleep(5000);`", to allow the page to load before performing each action. After much configuring and trial and error, switching between xPath and CSS

Selector, I couldn't get the selectors to work correctly. This resulted in the tests failing. Unfortunately, due to the structure of the Bot Framework, I had to limit my testing to the Bot Framework Emulator and manual test cases. Figure 37 below shows my UnitTest class.

```
namespace BotTest
{

    [TestClass]
    0 references
    public class TestCase
    {
        private IWebDriver driver;
        private string baseURL;
        private bool acceptNextAlert = true;


        [TestMethod]
        0 references
        public void TheCaseTest()
        {

            driver = new FirefoxDriver();
            baseURL = "https://karlfyp.azurewebsites.net/";

            driver.Navigate().GoToUrl(baseURL + "/Home/Chatbots");




            Thread.Sleep(5000);
            //driver.FindElement(By.CssSelector("input.wc-shellinput")).Clear();
            driver.FindElement(By.XPath("//div[@id='BotChatElement']/div/div/div[3]/div/input")).Clear();
            Thread.Sleep(2000);
            driver.FindElement(By.CssSelector("input.wc-shellinput")).SendKeys("hello");

            Thread.Sleep(2000);
            driver.FindElement(By.CssSelector("label.wc-send > svg > path")).Click();
            // driver.FindElement(By.XPath("//div[@id='BotChatElement']/div/div/div[3]/label[2]")).Click();
        }
    }
```

FIGURE 37 UNIT TEST CLASS

Conclusions

Overall I feel that working on this project provided me with a great learning experience. This has been my first major college project that was completely solely by me so it taught me a lot in terms of time management and work load management. My favourite part of this project was working on an area of computing that is really starting to grow and break into the mainstream market. With so many companies getting on board by creating their own chatbots, in a few years we will see it as an everyday thing. It has been great to work with Microsofts Bot Framework that is still in development and more and more features being added each day. Regrettably, I feel that I did spend a lot of time working with AIML before I moved on to Bot Framework, I feel that if I had have been working with Bot Framework from the start, I would've achieved a lot more. The feedback from Keith and Eugene at my mid-point presentation really set me in the right direction. In the future, I will remember to be realistic with the goals I set myself and to explore every available option to me before committing to one framework.

Although this project was great to work on, I'm disappointed that I did not get to implement any of the security features that I have learned about in my other modules completed in semester 2. I've learned a lot in the security aspect of computing from my other modules but it has been a learning experience developing this application as it touched on areas that I've never worked with before which gave me a challenging but beneficial experience.

Further development

With Microsoft Bot Framework still only being in the preview phase, many more features are still to come which I will be keeping a close eye on. Since I've started working with this Framework more features have been added and more and more

companies are starting to add their own bots to the bot directory on the Microsoft website.

One opportunity for this project to expand could be an official NCI Student Union chatbot. I feel that it would be an interesting addition to the Student Union Facebook page and could provide students with a quick and straightforward way to get answers for their queries. For example, the Faq feature of my NCI bot could be implemented to take care of the general queries. Users could view timetables completely within their Messenger app, an application that I'm sure every college student will have. Another function that could be of use is a form builder, which was used in the Pizza Ordering chatbot. Forms could be created for students to fill that could be done completely in the Messenger app or with the web client. These could be personal circumstance forms or other types of forms needed by the college. I believe that the college could make use of the computing students and work together to create something that could benefit both the students and the college.

From researching into the area of intelligent chatbots, I've noticed that there are a lot of startups setting up with the objective of creating chatbots for businesses and organizations, allowing their customers to tailor the chatbot to their needs. I feel that it would be worth my while and follow this scene as my experience creating this project could give me the extra edge I need to get involved in one of these startups. For example, I came across a startup with their office located in Berlin, (Which I am moving to in June) called Job Pal. They specialise in creating chatbots that will interview potential candidates for a job. They plan to automate the recruitment screening process and hand if off to bots that can interview candidates 24/7 on platforms that they know and trust. It cuts out the time wasted on screening irrelevant candidates. The company is hiring for an intern position which I have applied for. I emailed them not too long ago and managed to get a visit to the office to chat with one of their employees once I get over to Berlin.

References

Anon, (2016). [online] Available at: https://www.chatbots.org/ai_zone/ [Accessed 11 Dec. 2016].

Foundation, A. (2016). AIML - The Artificial Intelligence Markup Language. [online] Alicebot.org. Available at: http://www.alicebot.org/aiml.html [Accessed 11 Dec. 2016].

Pandorabots.com. (2016). The Slashdot Interview. [online] Available at: http://www.pandorabots.com/pandora/pics/wallaceaimltutorial.html [Accessed 11 Dec. 2016].

Vignesh Jayapalan, H. (2016). Android — Baking a simple ChatBot in 30 minutes (AIML). [online] Medium. Available at: https://medium.com/@harivigneshjayapalan/android-baking-a-simple-chatbot-in-30-minutes-aiml-ff43c3269025#.q9zghn2tl [Accessed 11 Dec. 2016].

Wallace, R. (2016). AIML 2.0 Working Draft. [online] Docs.google.com. Available at: https://docs.google.com/document/d/1wNT25hJRyupcG51aO89UcQEiG-HkXRXusukADpFnDs4/pub [Accessed 11 Dec. 2016].

www.tutorialspoint.com. (2016). AIML Tutorial. [online] Available at: http://dev.tutorialspoint.com/aiml/index.htm [Accessed 11 Dec. 2016].

Turing, A. (1950). Computing Machinery and Intelligence. 1st ed. Mind.

Ankitbko.github.io. (2017). *Chatbot using Microsoft Bot Framework - Part 4 · F5*. [online] Available at: http://ankitbko.github.io/2016/09/ChatBot-using-Microsoft-Bot-Framework-Part-4/ [Accessed 26 April 2017].

Anon, (2017). [online] Available at: http://streamcode.io/luis-in-depth/ [Accessed 4 April 2017].

Anon, (2017). *Speed Is A Killer - Why Decreasing Page Load Time Can Drastically Increase Conversions*. [online] Available at: https://blog.kissmetrics.com/speed-is-a-killer/ [Accessed 14 April 2017].

Blog.pal.chat. (2017). *Everybody is ready for chatbots — is Facebook?*. [online] Available at: https://blog.pal.chat/post/everybody-is-ready-for-chatbots-is-facebook [Accessed 1 May 2017].

Channel 9. (2017). *Conversational UI using the Microsoft Bot Framework*. [online] Available at: https://channel9.msdn.com/Events/TechDaysOnline/MVP-Led-

TechDays-Online-February-2017/Conversational-UI-using-the-Microsoft-Bot-Framework [Accessed 7 May 2017].

Code.msdn.microsoft.com. (2017). *Image Caption Bot using Microsoft Bot Framework and Cognitive Services [C#] in C# for Visual Studio 2015*. [online] Available at: https://code.msdn.microsoft.com/Image-Caption-Bot-using-61974c49 [Accessed 1 April 2017].

Docs.botframework.com. (2017). *Getting Started | Bot Builder SDK C# Reference Library | Bot Framework*. [online] Available at: https://docs.botframework.com/en-us/csharp/builder/sdkreference/ [Accessed 12 February 2017].

Gary Pretty. (2017). *Adding rich attachments to your QnAMaker bot responses*. [online] Available at: http://www.garypretty.co.uk/2017/02/23/adding-rich-attachments-to-your-qnamaker-bot-responses/ [Accessed 3 April 2017].

Gary Pretty. (2017). *BestMatchDialog for Microsoft Bot Framework now available via Nuget*. [online] Available at: http://www.garypretty.co.uk/2016/08/01/bestmatchdialog-for-microsoft-bot-framework-now-available-via-nuget/ [Accessed 3 April 2017].

Gary Pretty. (2017). *Building conversational forms with FormFlow and Microsoft Bot Framework – Part 2 – Customising your form*. [online] Available at: http://www.garypretty.co.uk/2017/01/10/building-conversational-forms-with-formflow-and-microsoft-bot-framework-part-2-customising-formflow/ [3 April 2017].

Gary Pretty. (2017). *Building conversational forms with FormFlow and Microsoft Bot Framework – Part 1*. [online] Available at: http://www.garypretty.co.uk/2017/01/09/building-conversational-forms-with-formflow-and-microsoft-bot-framework-part-1/ [Accessed 3 April 2017].

Gary Pretty. (2017). *Creating your first bot with the Microsoft Bot Framework – Part 1 – Build and test locally*. [online] Available at: http://www.garypretty.co.uk/2016/07/14/creating-your-first-bot-with-the-microsoft-bot-framework-part-1/ [Accessed 3 April 2017].

Docs.botframework.com. (2017). Dialogs | Bot Builder SDK C# Reference Library | Bot Framework. [online] Available at: https://docs.botframework.com/en-us/csharp/builder/sdkreference/dialogs.html [Accessed 3 April 2017].

Docs.botframework.com. (2017). FormFlow | Bot Builder SDK C# Reference Library | Bot Framework. [online] Available at: https://docs.botframework.com/en-us/csharp/builder/sdkreference/forms.html [Accessed 3 April 2017].

GitHub. (2017). Microsoft/BotBuilder. [online] Available at: https://github.com/Microsoft/BotBuilder [Accessed 15 March 2017].

Gary Pretty. (2017). *Forwarding activities / messages to other dialogs in Microsoft Bot Framework*. [online] Available at: http://www.garypretty.co.uk/2017/03/26/forwarding-activities-messages-to-other-dialogs-in-microsoft-bot-framework/ [Accessed 3 April 2017].

Gary Pretty. (2017). *Using Scorables for global message handling and interrupting dialogs in Bot Framework*. [online] Available at: http://www.garypretty.co.uk/2017/04/13/using-scorables-for-global-message-handling-and-interrupt-dialogs-in-bot-framework/ [Accessed 3 April 2017].

Hargan, J. and Hargan, J. (2017). The Death of the SMS?. [online] KillBiller. Available at: http://www.killbiller.com/blog/the-decline-of-the-sms [Accessed 4 May 2017].

Gatti, F. (2017). *Bots shift towards AI and garner $24 billion of investment*. [online] VentureBeat. Available at: https://venturebeat.com/2017/03/27/bots-shift-towards-ai-and-garner-24-billion-of-investment/ [Accessed 2 May 2017].

Osborne, R. (2017). *Botframework Web Chat Embedding | Robin Osborne*. [online] Robinosborne.co.uk. Available at: https://www.robinosborne.co.uk/2016/07/25/botframework-web-chat-embedding/ [Accessed 27 March 2017].

Osborne, R. (2017). *Create your first botframework bot in Azure! | Robin Osborne*. [online] Robinosborne.co.uk. Available at: https://www.robinosborne.co.uk/2016/07/12/create-your-first-botframework-bot-in-azure/ [Accessed 27 March 2017].

Osborne, R. (2017). *Create your first QnA bot using botframework's QnA Maker | Robin Osborne*. [online] Robinosborne.co.uk. Available at: https://www.robinosborne.co.uk/2016/09/26/create-your-first-qna-bot-using-botframeworks-qna-maker/ [Accessed 27 March 2017].

Osborne, R. (2017). *Implementing LUIS Routing within BotFramework | Robin Osborne*. [online] Robinosborne.co.uk. Available at: https://www.robinosborne.co.uk/2016/10/28/implementing-luis-routing-within-botframework/ [Accessed 27 March 2017].

Oswalt, I. (2017). *What Will a Chatbot Cost Me - And Is It Worth It?*. [online] Blog.21handshake.com. Available at: http://blog.21handshake.com/what-will-a-chatbot-cost-me-and-is-it-worth-it [Accessed 28 April 2017].

Appendix

Project Proposal

Project Proposal

Intelligent Chatbot with Google

Karl Lyons

X13452508

Karl.Lyons@student.ncirl.ie

BSc (Hons) in Computing

Cyber Security

Date

Objectives

To create an intelligent AIML based chat bot that can allow a human interacting with the bot to have an ongoing, interesting and enriched conversation featuring looked up information from Google.

Main Objectives

- Basic functionality – ability to respond to basic words/phrases
- Advanced logic – be able to talk about different topics
- Pull data from Google. E.g What's the weather like etc, current news events
- The bot should be able to pick topics to talk about rather than waiting on user input

Background

Most Chat Bots imitate conversation in a usually rather empty matter and the conversation lacks any substance. With basic chat bots the conversation is never really flowing, the bot always relies on the user to bring up topics which can make the conversation feel like a chore. An inspiration for this project would be Apples "Siri". Although Siri isn't really a chat bot it has the ability to search Google for you and display data based on those searches. For example, football results from last night.

Technical Approach

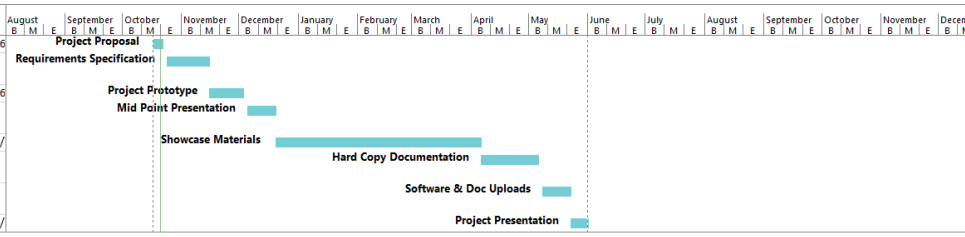
The application will be developed using AIML 2.0, it is an XML-compliant language that is used for creating natural language software agents. I will combine this with Java to run the application.

Special resources required

No special resources required.

Project Plan

ID	Task Mode	Task Name	Duration	Start	Finish
1		Project Proposal	5 days	Mon 17/10/	Fri 21/10/16
2		Requirements Specification	16 days	Mon 24/10/16	Mon 14/11/16
3		Project Prototype	14 days	Tue 15/11/1	Fri 02/12/16
4		Mid Point Presentation	11 days	Mon 05/12/16	Mon 19/12/16
5		Showcase Materials	77 days	Tue 20/12/1	Wed 05/04/
6		Hard Copy Documentation	22 days	Thu 06/04/17	Fri 05/05/17
7		Software & Doc Uploads	11 days	Mon 08/05/17	Mon 22/05/17
8		Project Presentatio	7 days	Tue 23/05/1	Wed 31/05/



Technical Details

The application will be written using AIML 2.0 to develop the responses to queries from the user. The AIML is interpreted by the java library ProgramAB and outputted to the android application. When search functionality is implemented the application will use Google to source its data. Other websites may come into play depending on what topic is being discussed in the conversation for example if the user asks the weather, the data could be pulled from a weather forecast website. The application will be able to store user details like Name, gender, age and possibly some other information depending on the conversation. This data will be stored for conversation enhancement and will be stored securely.

Evaluation

The system will be evaluated by test cases for each functional requirement.

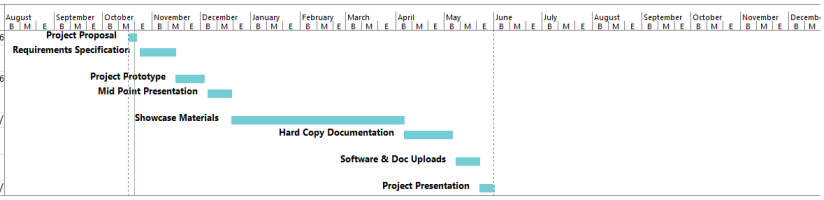
The system will be evaluated to the end user by having a mock conversation and testing each of the functional requirements.

Karl Lyons 21/10/16

Signature of student and date

Project Plan

ID	Task Mode	Task Name	Duration	Start	Finish
1		Project Proposal	5 days	Mon 17/10/	Fri 21/10/16
2		Requirements Specification	16 days	Mon 24/10/16	Mon 14/11/16
3		Project Prototype	14 days	Tue 15/11/1	Fri 02/12/16
4		Mid Point Presentation	11 days	Mon 05/12/16	Mon 19/12/16
5		Showcase Materials	77 days	Tue 20/12/1	Wed 05/04/
6		Hard Copy Documentation	22 days	Thu 06/04/17	Fri 05/05/17
7		Software & Doc Uploads	11 days	Mon 08/05/17	Mon 22/05/17
8		Project Presentatio	7 days	Tue 23/05/1	Wed 31/05/



Monthly Journals

September Journal

Student name: Karl Lyons

Programme : BSc Computing

Month: September 2016

Introduction

This month I wasn't able to decide on a final year project idea. I'm in the cyber security stream and wanted to do a project based around cyber security. I had a few different ideas but nothing solid that I wanted to commit to for a final year project. I waited for the list of projects from the different lecturers and the Intelligent Chat Bot project piqued my interest. I did some research into the idea and it seems like something I'd be interested in and will enjoy working on throughout the year.

I've emailed Ralf, the lecturer who proposed the idea, and asked him to meet for a chat about the project sometime next week.

Next month I'll be a making a start on the project and working away at that.

Supervisor Meetings

I don't have a supervisor yet so no meetings have been had yet.

October Journal

Student name: Karl Lyons

Programme : BSc Computing

Month: October 2016

Introduction

This month I met with Ralf and decided I was going to go ahead with the Intelligent Chat Bot project. I've been researching ways in which to do this and came across AIML 2.0 (Artificial Intelligence Markup Language) which is an XML-Compliant language. This is what I'm going to be building my project with. It's basically been created for making chat bots so I think it's perfect for my project. It's really easy to understand and get the hang of so I can get working on it straight away.

My Achievements

This month, I finished my project proposal and decided on a framework to build my project off.

My Reflection

I feel it kind of took me too long decide on a project when I could've been working on it earlier.

However, I have everything ready to go now so I can start working on it. I plan to get a decent amount of work done next month in November.

Intended Changes

Next month I will allocate more time to the project and try to balance the workload more evenly.

Supervisor Meetings

Date of Meeting: 21st October

Items discussed: Accepting the Project, Project Goals etc

Action Items: Just talking about what the project should be like and the end goals of the project.

November Journal

Student name: Karl Lyons

Programme : BSc Computing

Month: November 2016

Introduction

This month was tough as there was a lot of CA's and deliverables throughout the month so I didn't have as much time as I would have liked to work on the project. I decided to develop the project for Android as I couldn't figure out way to get a GUI working with the desktop version I had been working on. At the moment I'm working on getting up to speed with Android Studio as I have never worked with it before. It seems very similar to Java which I have worked with before so hopefully I will be able to get up to speed fairly fast.

My Achievements

Finished the Requirements Spec

My Reflection

I managed to get the chatbot onto android so I'm quite happy with that.

Intended Changes

Next month I need to put a lot more work into the project as it has been neglected for a while due to other modules taking priority, next semester I have less modules so I will be able to dedicate more time to the project.

December Journal

Student name: Karl Lyons

Programme : BSc Computing

Month: December 2016

Introduction

This month I was very busy with finishing off projects for my other modules. Ran into some difficulty with Android studio getting my AIML files to load up properly for my mid-point presentation. I had a command line chatbot running for my prototype which could show off what is possible with AIML.

My Achievements

Finished the Requirements Spec and Technical Report, although I was not as happy with it as I would like to be at least it is finished.

My Reflection

This month was tough because it was difficult to get a hold of my supervisor to review my technical report.

Intended Changes

Next semester the workload is cut in half so I will have a lot more time to work on the software project.

January Journal

Student name: Karl Lyons

Programme : BSc Computing

Month: January 2017

Introduction

This month was mostly spent studying for the semester 1 exams. I took on board the feedback from my mid point presentation and started to think about what I could do to improve my project. I explored different technologies as the one I was using at the moment wasn't giving me the freedom to do what I wanted with it. I came across Microsofts newly developed Bot Framework, which is pretty much designed for what I wanted to create. As well as Bot framework I found WIT.AI and API.AI, owned by Facebook and Google respectively.

My Reflection

I took Keith and Eugenes feedback and decided to make some changes with my project. Keith made a point of saying that I shouldn't try to create the next Siri or Alexa but instead try to work on different areas and showcase what is possible with chat bots. Although the advice was a bit blunt, I think I needed it as AIML just wasn't able to do what I needed and I feel that I wasted too much time messing around with that and it reflected in my grade for the mid-point presentation. Eugene gave me feedback relating to the security of chatbots so I will be keeping this in mind as get further into the development of my project. A lot of the security relies on the platforms my chatbot is hosted on so I'm not really sure how I can implement security into the chatbots themselves. I'm sure I will figure something out as time goes by. I will be taking this advice on board as I try to work with a new framework for my project.

Intended Changes

Next month I intend to explore bot framework, WIT.AI and API.AI and see what would best suit my needs.

February Journal

Student name: Karl Lyons

Programme : BSc Computing

Month: February 2017

Introduction

This month I explored Microsofts Bot Framework, WIT.AI, API.AI and tried to come up with some ways to use it with my project.

I decided to go with Microsofts Bot Framework mainly because of the documentation and that the platform mainly uses C#, a language that I am familiar with and have completed many projects with over the years. API.AI seemed to be more geared towards big development teams as a lot of the features were locked behind a paywall. Although it did have a lot of cool features such as support for smart homes and smart TVs, I just didn't really like the feel of it.

WIT.AI is another framework that I looked at that. It was similar to bot frameworks LUIS in that it looks to extract the intent from a users message. It had some nice features similar to API.AI like the smart home and smart TV stuff but most of its development was done with the web client which I wasn't a fan of. The languages that it support were Python, Ruby and Node.Js, all of which I am not familiar with and have never worked with before so that was a big no from me. I decided to go with Bot Framework as it suited me the best.

After a good while of working with Bot Framework, I managed to develop a Q&A chatbot using NCI's FAQ page online. The bot is deployed Facebook Messenger so it makes it really easy for me to test it and see how things are working. I'm using Microsoft Azure to host my project and it works really well. I can test out my code using an emulator on my PC and then when I think it's ready to publish to the live Facebook version it's simply a click away. Since the Framework is relatively new it is hard to find some help online with how to get certain features working but I'm sure I'll find a way.

My Achievements

Got my chatbot hosted with Azure and published onto Facebook Messenger with its own Page.

My Reflection

This was a good month for my project as I got a functional Chat bot up and running and hosted online. Looking forward to next month to get more work completed. Having the chatbot hosted online with Bot Framework is so much better for me compared to using AIML which I wasn't able to host online. AIML was holding me back I feel and I'm glad I switched over to Bot Framework. Really helps having it up online especially on Facebook so I can send it to friends to test out and give me some feedback.

Supervisor Meetings

Date of Meeting: 16th Feb

Items discussed: Chatbot, Microsoft Bot Framework, Bot functionality (twitter search, weather bot etc)

Action Items: More functionality.

March Journal

Student name: Karl Lyons

Programme : BSc Computing

Month: March 2017

Introduction

This month I was really able to get sunk into the project as I had a lot of my work from other modules finished. I worked a lot with Bot Framework and have started to use LUIS. LUIS is a natural language processor which is used to derive the intent from users messages. I can use this recognize the intent from the users message without looking for a certain phrase, it gives the users a good bit of flexibility with their phrasing. This helps steer the conversations in a way so I can give the users what they need quite easily. I've started working on a NCI bot. The one I have already has QnA features implemented and

I plan on adding a service to check the timetables. Could also do this for exam timetables I'll think about adding that in later.

My other chatbots are the image caption bot which I think is really cool. You take a picture and it will tell you what it thinks it is. It does this using Microsofts computer vision API. Another one is the Pizza ordering chatbot, it allows users to order a pizza entirely within the Messenger application and has some validation and message handling.

I came across a tutorial online about how to implement global message handling, this allows me to set a keyword that can cancel a conversation and return it to the root dialog. This comes in handy if the user gets stuck or they typed the wrong message and they need to go back. It just adds more flexibility to the bots and makes them a bit easier to use.

Another feature I got in was the option to give feedback, I combined this with global message handling so when the user types feedback they will be taken to a form to submit user feedback. I used SendGrid API to send the emails. I only have a trial account so hopefully that doesn't cause any issues.

My Achievements

Developed the pizza bot and image caption bot and some other small ones but not fully functioning just yet. I'm starting to get the hang of bot framework which is great.

My Reflection

This month was great as I got a lot of work done on the project. Bot framework has been adding new features and some of the guys working on the development of it are putting out tutorials which have been really helpful for me as some of the documentation is outdated or just doesn't really provide what I need. I reckon in the next year or two bot framework will have so much more features and a lot more tutorials on how to add functionality. Although it is hard working with this framework as it is still in the early

stages of development, it's been fun and interesting to develop my project on this framework as it grows and grows every day.

Supervisor Meetings

Date of Meeting: 10th March

Items discussed: Chatbot, Microsoft Bot Framework, Bot functionality (Image Caption, NCI Bot

Action Items: Make the bot more responsive, more features