DCU Personal Assistant Chat Bot

Student: Ian Kelly (13480138)

Supervisor: Gareth Jones

Technical Guide

Abstract

The DCU Personal Assistant Chat Bot is a web app designed with the purpose of aiding DCU students. The student will be able to ask the bot a question, the bot will utilize natural language processing methods to determine what the student is asking, and then respond with the appropriate information. This information will include but is not limited to the time of their next lecture and when their next bus will arrive.

Table of Contents

Motivation

The idea to create a chat bot was brought about after 4 years of difficulty navigating the DCU website. The site is slow, difficult to navigate, and performs poorly on mobile devices. By this point it is tolerate. However, for new students joining the college this should not be an issue. The creation of the DCU chat bot was aimed at eliminating this by allowing the bot directly information from the DCU website and the bot fetches it for them. The bot idea then grew into a full personal assistant for students. The can ask about information on the DCU campus without having to open a map, and they can ask for bus times without ever having to download the Dublin Bus app or visiting the website.

The main goal of the chat bot is ease of use. The user simply must ask a question, maybe give a small amount of detail, and the bot does all the heavy lifting.

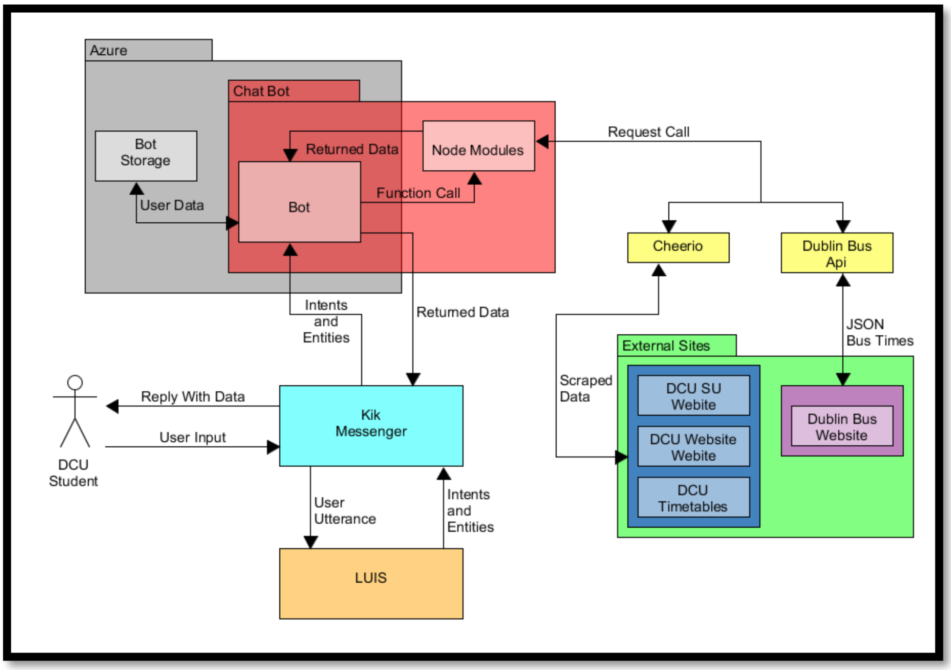
Research

With an idea that I wanted to make some sort of chatbot, I was directed to a git repository by a work colleague which was a hub of information for all things bot related. On there I found out about Microsoft's new software called LUIS. LUIS was a natural language processor that would allow me to create a chatbot with some intelligence. The LUIS website has plenty of documentation on starter guides to work with and through that I found Microsoft's Bot Builder. Using the Bot Builder, I can create a channel of data from between my bot, LUIS, and a chat interface.

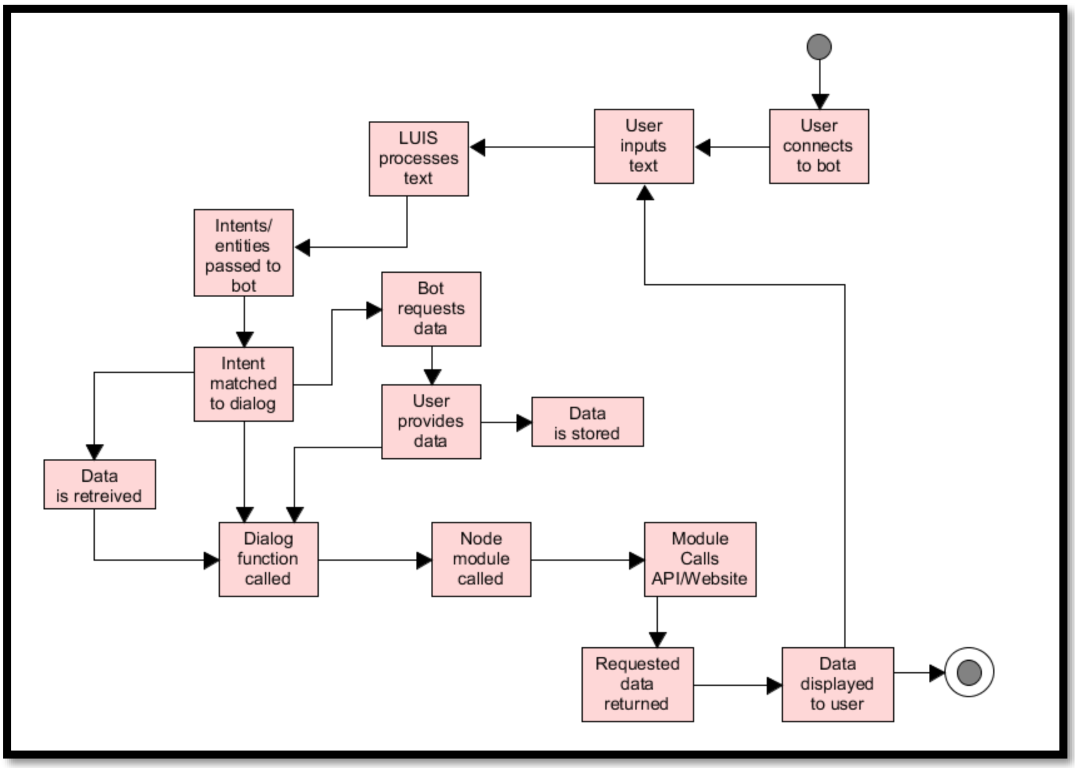
While researching I found a node module specifically for calling the Dublin Bus API found here. At the very beginning of my project and when just starting to work with node I used this as a point of reference of how a module fits together and how an API is called. The functions of this module return unparsed bus information so were not used but instead built off of.

Design

Below is a diagram show how the entire system interacts. The general design has not changed much since the functional specification. The user will send an input to Kik messenger, which will then send it to LUIS. LUIS will process the input and extract intents and entities. It will then send these to the main bot where the intent will be matched to a dialog. The dialog may ask for user data and will store this data on a table in azure. The matched dialog will then call a node module. The node module will then use either cheerio or the Dublin Bus API to retrieve and return data from an external website. This data will then be formatted to text and returned to the user.



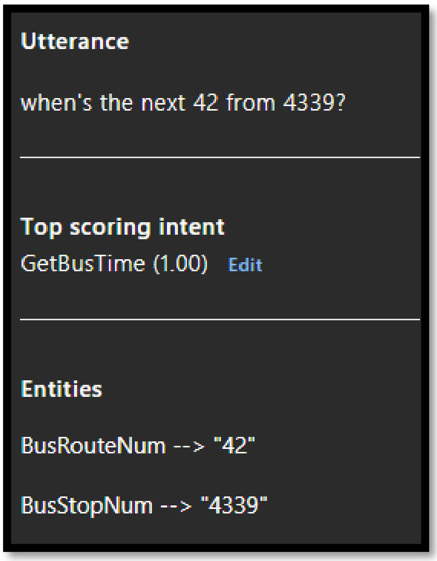
Below is an activity diagram how the steps taken from when the user asks a question to when they receive a response.



Implementation

LUIS

LUIS is implemented and tested on 'https://eu.luis.ai'. Once the LUIS model is built you can train and test the model by giving it user utterances



Utterance

when's the next 42 from 4339?

Top scoring intent
GetBusTime (1.00) Edit

Entities

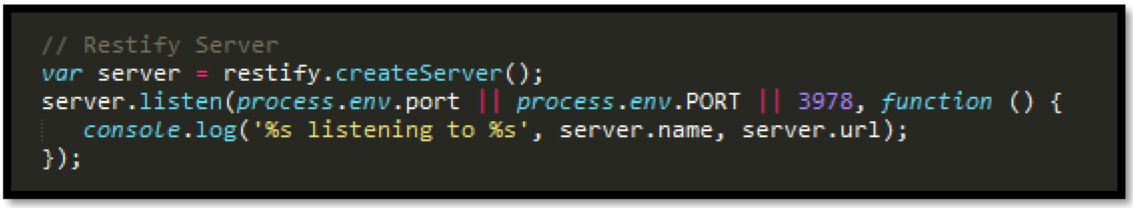
BusRouteNum --> "42"

BusStopNum --> "4339"

This testing can be alleviated by including batch testing. The batch file is a file containing several JSON utterances that can be passed to the model all at once. The batch file is updated continuously over the course of development to ensure the model has all user questions possibilities covered. Once trained, the LUIS model is published to a HTTP endpoint. The model can now be connected to the chat bot using the endpoint key. Utterances can then be passed from the bot as HTTP requests using the LUIS endpoint API and a JSON object will be returned with the results.

Bot Builder

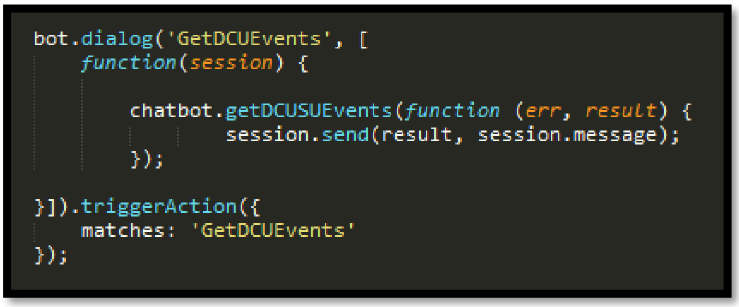
The chat bot is made using Microsoft's Bot Builder SDK using Node.js which acts as the connector for the bot and the conversation channel (Kik). Bot Builder uses Restify, a framework for web services to create a web server that will allow the chat bot to run on an API endpoint which will be hosted on Microsoft Azure. The documentation on the Bot Builder's site is very helpful and provides the basics to get started. When creating you bot on Azure much of this code is generated for you, which makes things easier to keep track of and all that's left is to connect it up properly.



```
// Restify Server
var server = restify.createServer();
server.listen(process.env.port || process.env.PORT || 3978, function () {
    console.log('%s listening to %s', server.name, server.url);
});
```

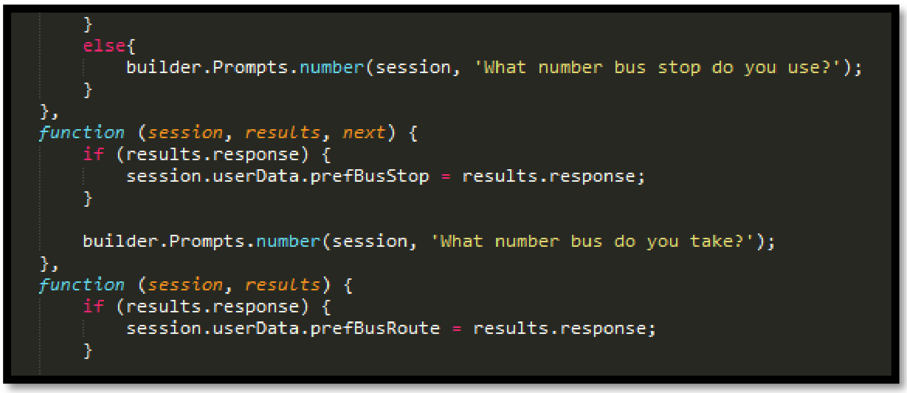
Bot Dialog

The chat bot's conversational functions are created using dialogs. The bot receives the intent from LUIS and tries to find a dialog that matches. When a dialog is found, that dialog contains functions that will return data to the user.



```
bot.dialog('GetDCUEvents', [
    function(session) {

        chatbot.getDCUSUEvents(function (err, result) {
                session.send(result, session.message);
        });

}]).triggerAction({
    matches: 'GetDCUEvents'
});
```

The user might say something like "What's going on in DCU?" and LUIS will recognize that as the 'GetDCUEvents' intent and will call this dialog. There are also some features of dialogs to add a sense of bot intelligence and conversation flow. Some dialogs contain prompt function that reply to the user and ask for some additional information. Their reply will also be stored on Azure and user data.



```
    }
    else{
        builder.Prompts.number(session, 'What number bus stop do you use?');
    }
},
function (session, results, next) {
    if (results.response) {
        session.userData.prefBusStop = results.response;
    }

    builder.Prompts.number(session, 'What number bus do you take?');
},
function (session, results) {
    if (results.response) {
        session.userData.prefBusRoute = results.response;
    }
```

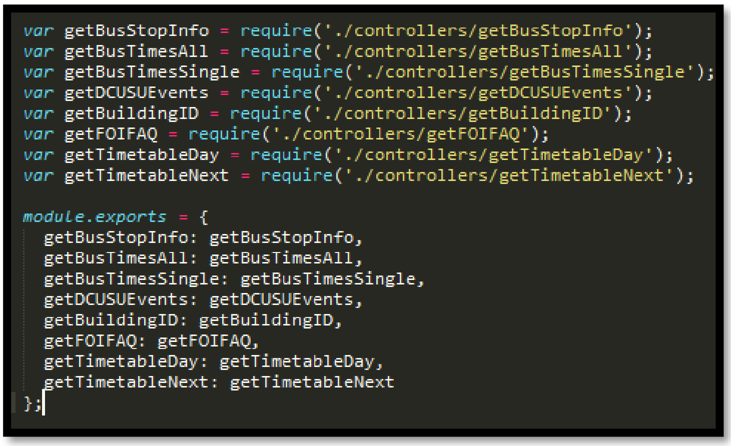
Here you can see the user is asked for their bus stop number. That value is then stored in session.userData.prefBusStop which will persist after the conversation has ended . Now the next time the user calls this intent they will not be asked for this information again. The user can change this data later.

Kik

Connection the chat bot to Kik was very simple and only required registering my chat bot on the Kik website and connecting my chat bot to Kik using the bot's endpoint key.

Node Modules

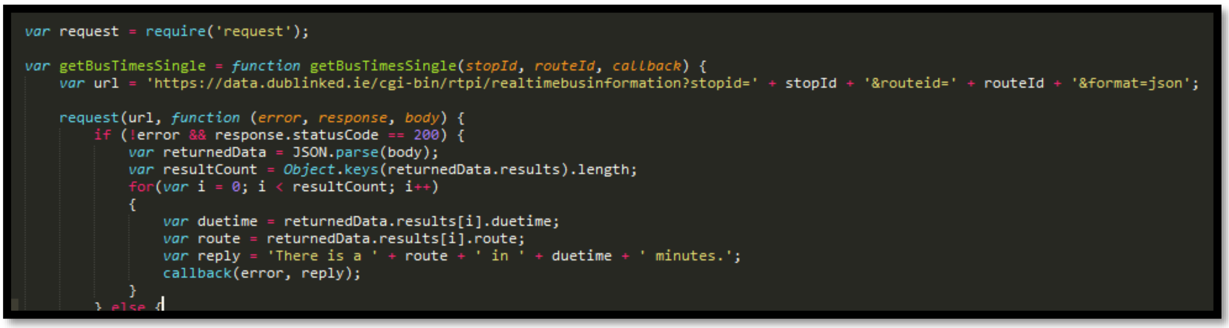
All of the functions for the chat bot are created in a Node.js node module and are exported to the main bot.



```
var getBusStopInfo = require('./controllers/getBusStopInfo');
var getBusTimesAll = require('./controllers/getBusTimesAll');
var getBusTimesSingle = require('./controllers/getBusTimesSingle');
var getDCUSUEvents = require('./controllers/getDCUSUEvents');
var getBuildingID = require('./controllers/getBuildingID');
var getFOIFAQ = require('./controllers/getFOIFAQ');
var getTimetableDay = require('./controllers/getTimetableDay');
var getTimetableNext = require('./controllers/getTimetableNext');

module.exports = {
  getBusStopInfo: getBusStopInfo,
  getBusTimesAll: getBusTimesAll,
  getBusTimesSingle: getBusTimesSingle,
  getDCUSUEvents: getDCUSUEvents,
  getBuildingID: getBuildingID,
  getFOIFAQ: getFOIFAQ,
  getTimetableDay: getTimetableDay,
  getTimetableNext: getTimetableNext
};
```

Dublin Bus

There are three modules for getting Dublin Bus information. One that gets the real time information for all times at a stop, one that gets times for a stop for a specific bus route, and one that gets the information about a bus stop. All were made by calling the Dublin Bus API located here. The bus time and bus route specified as entities from the user are sent to the module and a request call is made and a JSON object on the results is returned. The results are then parsed and then the results are exported to the bot using a callback.

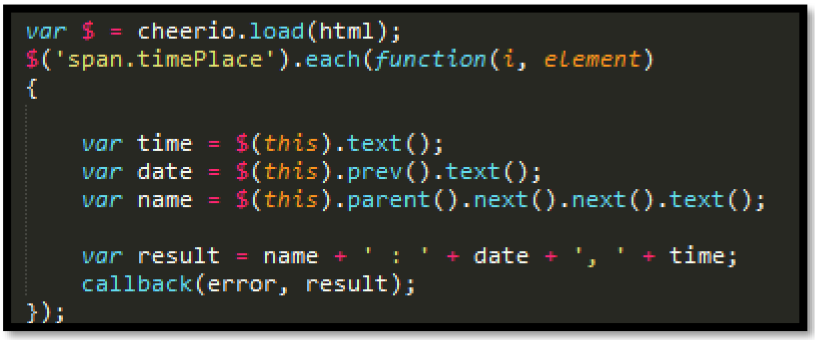


```
var request = require('request');

var getBusTimesSingle = function getBusTimesSingle(stopId, routeId, callback) {
    var url = 'https://data.dublinked.ie/cgi-bin/rtpi/realtimebusinformation?stopid=' + stopId + '&routeid=' + routeId + '&format=json';

    request(url, function (error, response, body) {
        if (!error && response.statusCode == 200) {
            var returnedData = JSON.parse(body);
            var resultCount = Object.keys(returnedData.results).length;
            for(var i = 0; i < resultCount; i++)
            {
                var duetime = returnedData.results[i].duetime;
                var route = returnedData.results[i].route;
                var reply = 'There is a ' + route + ' in ' + duetime + ' minutes.';
                callback(error, reply);
            }
        } else {
```

This turned out to be one of the simpler parts of the project due to its short length. However, this was the first module that I worked on and the challenge for this was working out how a module works, how to call an API, how to parse JSON for the data I needed, and how to export this data correctly back to the bot. Figuring out how to data from a JSON body took me longer than I would like to mention.

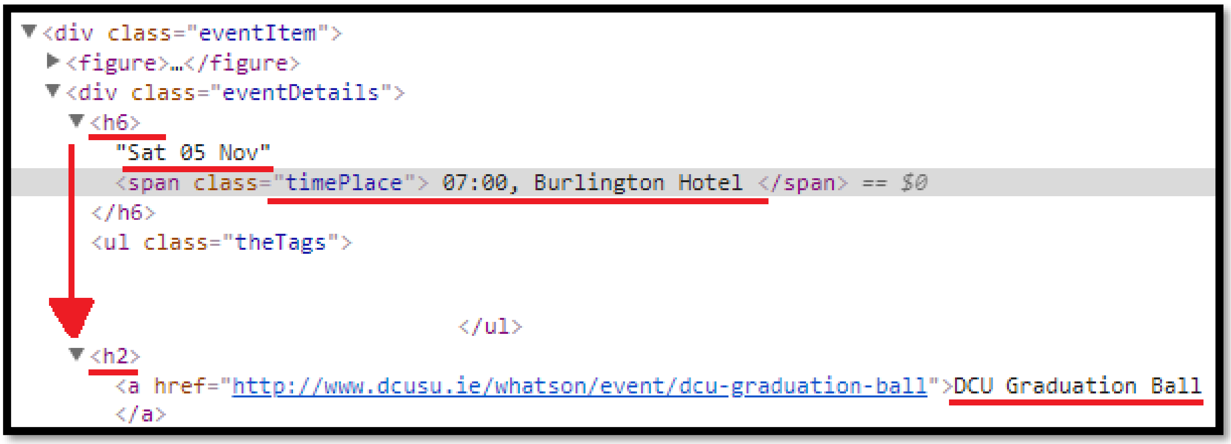
DCU Events

For the rest of the modules, whenever data needs to be scraped from a site and there is no API, a node module named Cheerio will be used. Cheerio is essentially just a core subset of jQuery for server use that I can use with Node.js. I can use jQuery to traverse the HTML of a page and collect the data that I need. I scraped the events for DCU from the Student Union's website 'http://www.dcusu.ie/whatson/'.



```
var $ = cheerio.load(html);
$('span.timePlace').each(function(i, element)
{

    var time = $(this).text();
    var date = $(this).prev().text();
    var name = $(this).parent().next().next().text();

    var result = name + ' : ' + date + ', ' + time;
    callback(error, result);
});
```

This is a loop for every time the 'timePlace' attribute appears. This was chosen because it is a unique attribute for each event, so I can scrape all events on the page without scrape unwanted data.

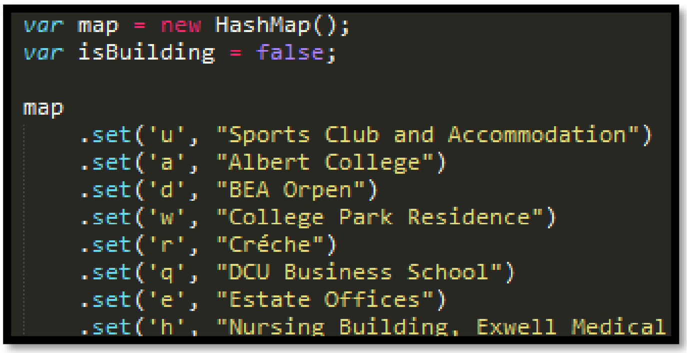


```
▼<div class="eventItem">
  ▶<figure>…</figure>
  ▼<div class="eventDetails">
    ▼<h6>
       "Sat 05 Nov"
       <span class="timePlace"> 07:00, Burlington Hotel </span> == $0
     </h6>
     <ul class="theTags">


                                   </ul>
    ▼<h2>
       <a href="http://www.dcusu.ie/whatson/event/dcu-graduation-ball">DCU Graduation Ball
       </a>
```

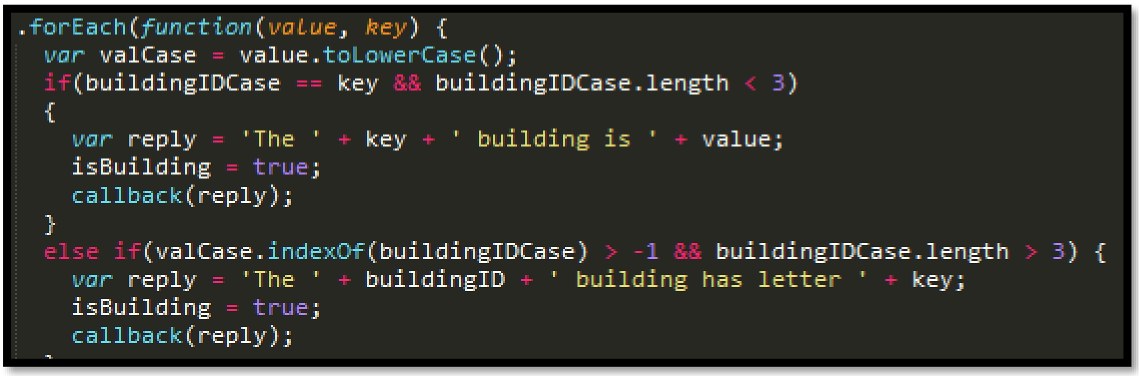
As you can see I start at timePlace and get the text of this element to find the time of the event. I then move up on element and get the text of the date of the event. I then move to the parent of these elements and down 2 places. I then get the text of this element to find the name of the event.

DCU Campus

There are a number of modules for general DCU campus information. There are modules for finding bike stations, campus locations, and the names and ID of specific buildings. These modules do not call external sites and contain hardcoded responses to user questions as these responses will always be the same (The response to asking where Nubar is will always be the same). Finding the name or ID of a building does have some dynamic functions however. When the intent is called, it takes the entity from the utterance and checks whether it is an ID or a building name. It then goes through a hashmap where the keys are the building IDs and the values are the building name. If the entity matches a key, it returns a value. If the entity matches a value it returns a key.



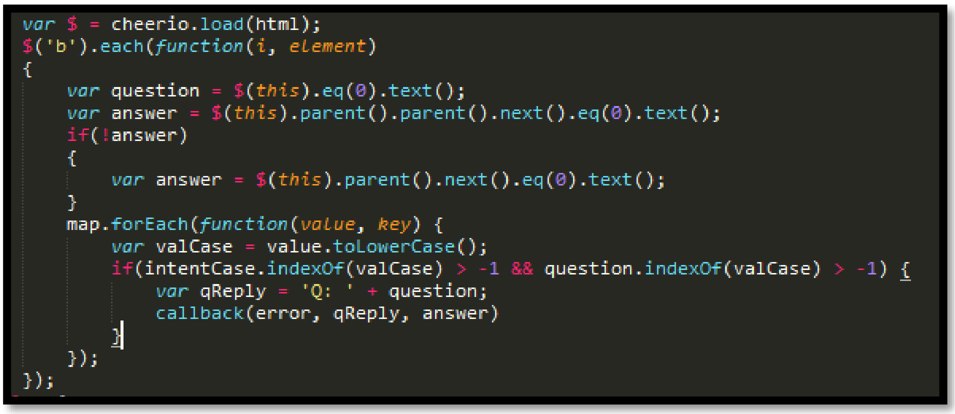
```
var map = new HashMap();
var isBuilding = false;

map
    .set('u', "Sports Club and Accommodation")
    .set('a', "Albert College")
    .set('d', "BEA Orpen")
    .set('w', "College Park Residence")
    .set('r', "Créche")
    .set('q', "DCU Business School")
    .set('e', "Estate Offices")
    .set('h', "Nursing Building, Exwell Medical
```



```
.forEach(function(value, key) {
  var valCase = value.toLowerCase();
  if(buildingIDCase == key && buildingIDCase.length < 3)
  {
    var reply = 'The ' + key + ' building is ' + value;
    isBuilding = true;
    callback(reply);
  }
  else if(valCase.indexOf(buildingIDCase) > -1 && buildingIDCase.length > 3) {
    var reply = 'The ' + buildingID + ' building has letter ' + key;
    isBuilding = true;
    callback(reply);
```

DCU FAQ

Frequently asked questions on the DCU website are scraped using jQuery. The FAQs that can be asked in the chat bot are for the Freedom of Information Office 'https://www.dcu.ie/foi/faq.shtml'. Each question and its answer are scraped. Then module then takes the user question and checks to see if keywords from each question are in the user's question. If they are, the question and answer are returned.

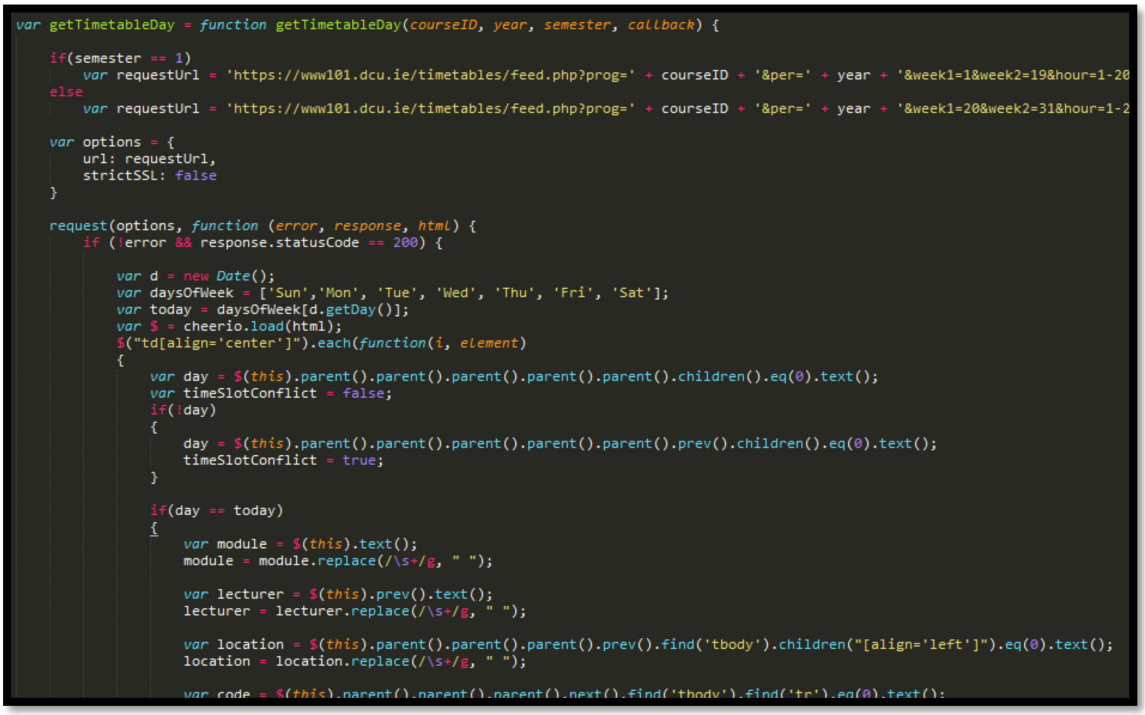


```
var $ = cheerio.load(html);
$('b').each(function(i, element)
{
    var question = $(this).eq(0).text();
    var answer = $(this).parent().parent().next().eq(0).text();
    if(!answer)
    {
        var answer = $(this).parent().next().eq(0).text();
    }
    map.forEach(function(value, key) {
        var valCase = value.toLowerCase();
        if(intentCase.indexOf(valCase) > -1 && question.indexOf(valCase) > -1) {
            var qReply = 'Q: ' + question;
            callback(error, qReply, answer)
        }
    });
});
```

One of the main challenges with scraping these is ensure that that data that is scraped and returned is accurate. This involves planning how the HTML is traversed.

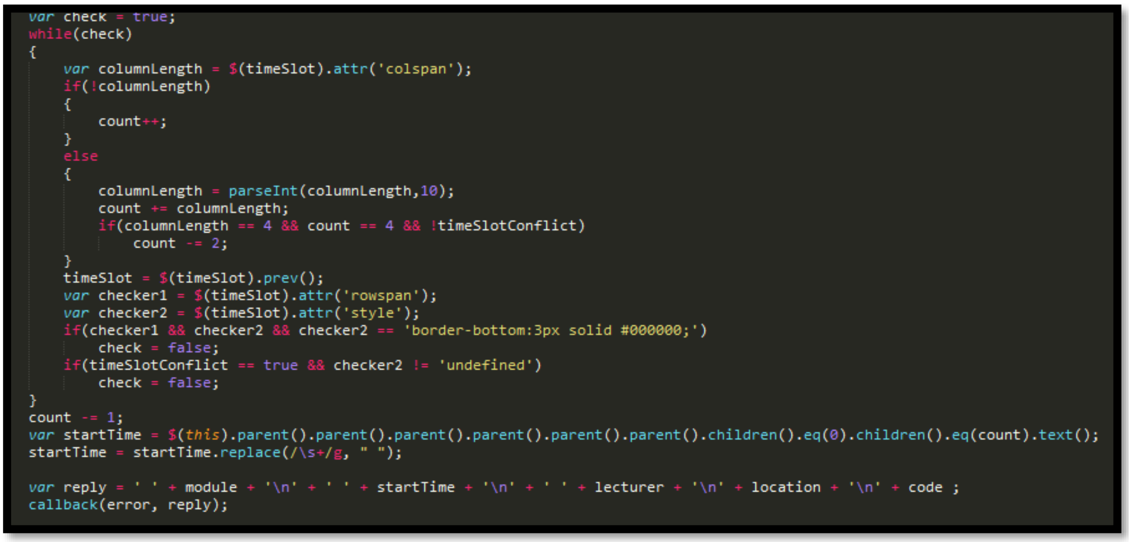
DCU Timetables

There are 2 timetable modules, one for getting all lectures for the day and another for finding the user's next lecture. The DCU timetable modules are scraped from 'https://www101.dcu.ie' using jQuery. The site has a certificate issue so strictSSL needs to be set to false in order to access the site. The module first takes the user data to find the right timetable URL. It then finds what day of the week it is and scrapes the data for each lecture from the HTML.



```javascript
var getTimetableDay = function getTimetableDay(courseID, year, semester, callback) {

    if(semester == 1)
        var requestUrl = 'https://www101.dcu.ie/timetables/feed.php?prog=' + courseID + '&per=' + year + '&week1=1&week2=19&hour=1-20
    else
        var requestUrl = 'https://www101.dcu.ie/timetables/feed.php?prog=' + courseID + '&per=' + year + '&week1=20&week2=31&hour=1-2

    var options = {
        url: requestUrl,
        strictSSL: false
    }

    request(options, function (error, response, html) {
        if (!error && response.statusCode == 200) {

            var d = new Date();
            var daysOfWeek = ['Sun','Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat'];
            var today = daysOfWeek[d.getDay()];
            var $ = cheerio.load(html);
            $("td[align='center']").each(function(i, element)
            {
                var day = $(this).parent().parent().parent().parent().parent().children().eq(0).text();
                var timeSlotConflict = false;
                if(!day)
                {
                    day = $(this).parent().parent().parent().parent().parent().prev().children().eq(0).text();
                    timeSlotConflict = true;
                }

                if(day == today)
                {
                    var module = $(this).text();
                    module = module.replace(/\s+/g, " ");

                    var lecturer = $(this).prev().text();
                    lecturer = lecturer.replace(/\s+/g, " ");

                    var location = $(this).parent().parent().parent().prev().find('tbody').children("[align='left']").eq(0).text();
                    location = location.replace(/\s+/g, " ");

                    var code = $(this).parent().parent().parent().next().find('tbody').find('tr').eq(0).text();
```

As the times for the lectures are on a separate table, it then needs to count how many time slots there are before the lecture and checks that it has reached the top of the table, then moves up to the time table, and then counts that many across.



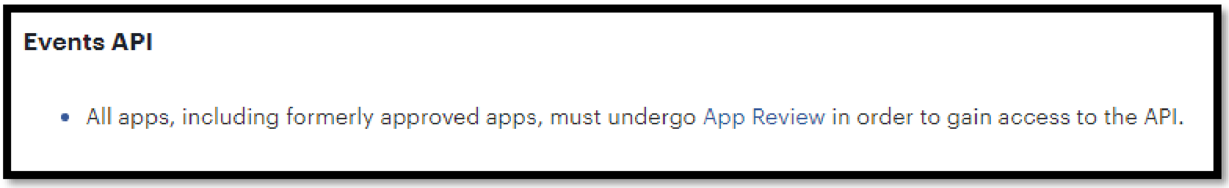
```javascript
var check = true;
while(check)
{
    var columnLength = $(timeSlot).attr('colspan');
    if(!columnLength)
    {
        count++;
    }
    else
    {
        columnLength = parseInt(columnLength,10);
        count += columnLength;
        if(columnLength == 4 && count == 4 && !timeSlotConflict)
            count -= 2;
    }
    timeSlot = $(timeSlot).prev();
    var checker1 = $(timeSlot).attr('rowspan');
    var checker2 = $(timeSlot).attr('style');
    if(checker1 && checker2 && checker2 == 'border-bottom:3px solid #000000;')
        check = false;
    if(timeSlotConflict == true && checker2 != 'undefined')
        check = false;
}
count -= 1;
var startTime = $(this).parent().parent().parent().parent().parent().parent().children().eq(0).children().eq(count).text();
startTime = startTime.replace(/\s+/g, " ");

var reply = ' ' + module + '\n' + ' ' + startTime + '\n' + ' ' + lecturer + '\n' + location + '\n' + code ;
callback(error, reply);
```

This was by far the most complex scrape and required a lot of planning to get right. There were also a number of problems that are mentioned below that need to be solved.

Problems Encountered

- Before this project I had never used any of the tools or languages involved. This meant that I had to learn JavaScript from scratch, I also had to learn Node.js and how node modules worked and interacted with me main bot. I had to learn how to call and get data from an API and how to parse JSON. I had never scraped from a website before and never used jQuery. This meant that there was a significant amount of time spent just figuring out what I was doing and how to do something.

- Creating a complete language model on LUIS. While the LUIS language model is not code-based there is still significant challenge in creating a model that works correctly. There are so many ways a user can ask a question and the model had to be able to handle multiple combination of utterances for the same intent. This however is entirely dependent on the combinations that I can think of. This issue can be solved in part by user testing and batch testing. However, LUIS can detect patterns among utterances of the same type. For example, if many utterances for the intent 'GetNextBus' contain the words 'next' and 'bus' in them, it is likely that what the user asks matches this intent if it contains the same words.
- Once I had some features working for the bot I could test them in a chat emulator. One issue was when there was a problem with one of the bot's functions that I would call from the emulator, the emulator would not produce error messages indicating what was wrong and would either send a message like "Oops, something went wrong, and we need to restart" or would not send any message at all. Also, for every change I made to the bot I would need to publish it to Azure and refresh the emulator, both of which would take a few minutes each. This meant that I would need to attempt to solve problems blind and then wait ~5 minutes between each attempt. This was a huge time sink and made initial progress on the bot extremely slow. Luckily, when I began making node modules I could test them in command prompt.
- For over a month I had issues with getting entities from LUIS. When I would ask my chat bot a question like "Give me the bus times from stop 4339", it would struggle to detect that 4339 was a bus stop number entity. Due to LUIS being a rather new technology there was minimal help online and for a lot time I had used hard-coded variables as a means of testing my bot. I eventually discovered that there was 2 reason I was having problems:
    1. The documentation on the LUIS site was slightly outdated and only used entities in if-statements like if(! stopNum) and did not mention that entities were returned as objects rather than the values of the entity. This was solved by calling the entity value like stopNum.entity. The documentation has since been updated.
    2. The entities for my model were not properly trained. The entities used in my utterances were of the same value and LUIS would fail to recognize entities of different lengths. For example, asking for the bus times for stop 4339 would work but asking for stop 23 would not due to it being of a different length. This was solved by the addition of phrase lists to my model. These were lists of words that LUIS would group together. A phrase list of all numbers from 1 to 10000 would train LUIS that these numbers can be interchangeable in user sentences.
- For the chat bot I had originally planned to host it on Facebook Messenger and also had planned to get the events of the DCU student union from their Facebook page using Facebook's Graph API. I had some of this work done, however in early April there was an update to the API which meant that any app that wanted to get event data must first submit their app for review along with a video and detailed explanation of the app's functions. Once it was submitted it would be approved in up to 2 weeks.

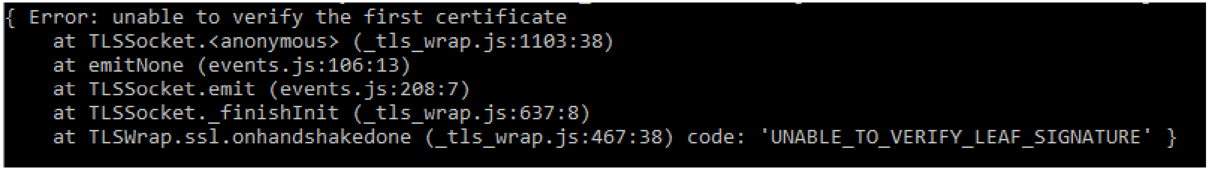


Events API

- All apps, including formerly approved apps, must undergo App Review in order to gain access to the API.

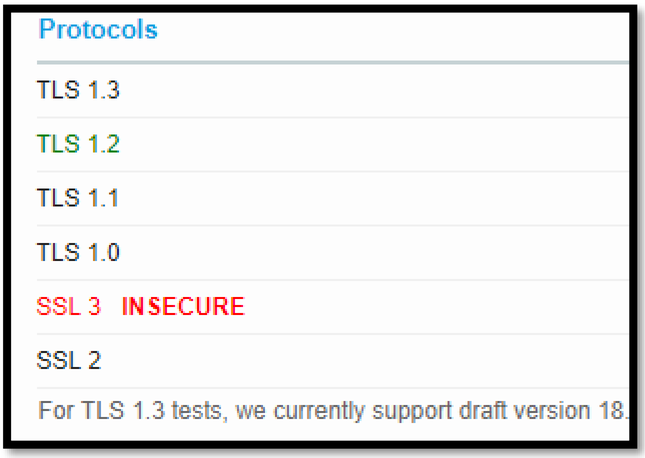
Since I did not have the time available to do this submission, my bot needed to be changed. The bot is now on Kik and events are now scraped for the DCU student union website. The only issue with this is that the events are less numerous and are updated less frequently.

- There seems to be an issue with dialog prompts in bot builder. When the bot prompts the user for a text reply, the bot seems to treat the reply as another utterance and cancels the current dialog. This means that it cannot correctly ask the user for their course code and as such cannot get their timetable. This issue is not present with prompting the user for a number response.
- I had been having an issue with accessing the 'www101.dcu.ie' website in order to scrape timetables. I was using the same method of scraping as I had with other websites however this time I was receiving an error saying there was a first certificate missing.



```
{ Error: unable to verify the first certificate
    at TLSSocket.<anonymous> (_tls_wrap.js:1103:38)
    at emitNone (events.js:106:13)
    at TLSSocket.emit (events.js:208:7)
    at TLSSocket._finishInit (_tls_wrap.js:637:8)
    at TLSWrap.ssl.onhandshakedone (_tls_wrap.js:467:38) code: 'UNABLE_TO_VERIFY_LEAF_SIGNATURE' }
```

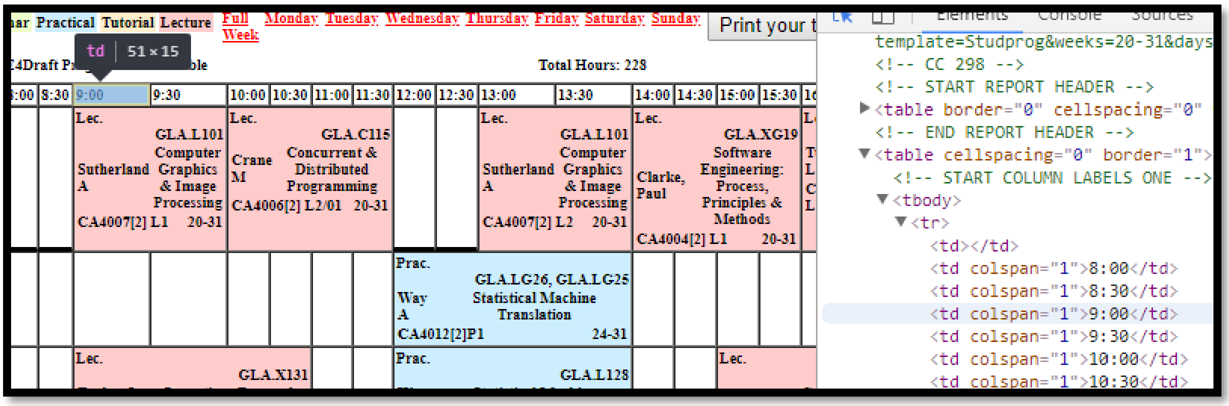
I contacted the ISS helpdesk in DCU via email and they assured me there was nothing wrong with the website and the issue was on my end. After running an SSL test on the site there does seem to be an issue.



Protocols
TLS 1.3
TLS 1.2
TLS 1.1
TLS 1.0
SSL 3 INSECURE
SSL 2
For TLS 1.3 tests, we currently support draft version 18.

This was resolved by setting strictSSL to false when calling the site.

- When Scraping timetables, the lectures and the times are separated into 2 different HTML tables. This means that the time of the lecture cannot be obtained the same way the other lecture's details (name, location) are obtained.



4Draft P ble Total Hours: 228

:00	8:30	9:00	9:30	10:00	10:30	11:00	11:30	12:00	12:30	13:00	13:30	14:00	14:30	15:00	15:30	16
		Lec. GLA.L101 Sutherland A Computer Graphics & Image Processing CA4007[2] L1 20-31		Lec. GLA.C115 Crane M Concurrent & Distributed Programming CA4006[2] L2/01 20-31						Lec. GLA.L101 Sutherland A Computer Graphics & Image Processing CA4007[2] L2 20-31		Lec. GLA.XG19 Clarke, Paul Software Engineering: Process, Principles & Methods CA4004[2] L1 20-31				L T L C L
								Prac. GLA.LG26, GLA.LG25 Way A Statistical Machine Translation CA4012[2]P1 24-31								
		Lec. GLA.X131						Prac. GLA.L128						Lec.		

```
template=Studprog&weeks=20-31&days
<!-- CC 298 -->
<!-- START REPORT HEADER -->
▶<table border="0" cellspacing="0"
<!-- END REPORT HEADER -->
▼<table cellspacing="0" border="1">
   <!-- START COLUMN LABELS ONE -->
  ▼<tbody>
    ▼<tr>
        <td></td>
        <td colspan="1">8:00</td>
        <td colspan="1">8:30</td>
        <td colspan="1">9:00</td>
        <td colspan="1">9:30</td>
        <td colspan="1">10:00</td>
        <td colspan="1">10:30</td>
```

This meant that to get the time I had to count the number of time slots before the lecture and count that amount along the time table.

- I got data from the DCU FAQ pages by scraping. This was less of a challenge as I had done this after scraping events. I chose not to add postgraduate and registry FAQ pages to the bot, not because they couldn't be scraped, but every single question had a different format (tables, bullet points etc.) and writing scraping code for 40 separate questions was a massive time sink that would not have been any new of a challenge. A challenge I did face for the FAQs however was how the bot would interpret which of the question on the page the user was asking. This was solved with a HashMap with key words for every question which would try match to what the user is asking. It is not perfect however and is reliant on the keyword being located in the user question.

Known Issues

- The chat bot is lacking features that would provide a complete suite of functions for DCU students. These would include finding water fountains/ATMs/bike racks on campus, more FAQ functions such as for postgraduate and registry, and library information. The reason these are not currently in the chatbot is because they are largely hardcoded responses or functions that would waste too much time being made and do not utilize and implementation methods that are not currently used in other functions.
- Another major issue with scraping timetables is that days with 2 lectures at the same time, the second row has no identifiable indicator as to when the timeslot count has reached the top of the table (Usually the day element is located there but is not present here). This means the count doesn't know when to stop and as such the times cannot be obtained. I have yet to find a viable solution to this.
- There has been infrequent stalling on the chatbot when results may only be partially shown. This appears to be an issue azure and LUIS returning data too slowly and overlapping.
- When the bot asks for course codes when finding timetable information, it can treat the text input as a user utterance and trigger another intent rather than following the current dialog. This is partially fixed by training LUIS to ignore certain user input as an utterance, however there may be cases that slip through.
- Dublin bus functionality does not current handle as just for a bus route and no stop, and does not handle referring to bus stop names rather than the numbers.
- There is some overlap with LUIS intents for intents that are similar in user utterance. This may cause some incorrect intents in some cases

- Some modules may return an empty string if there is no valid response. For example, getting your next class for the day after you last class has finished returns nothing.

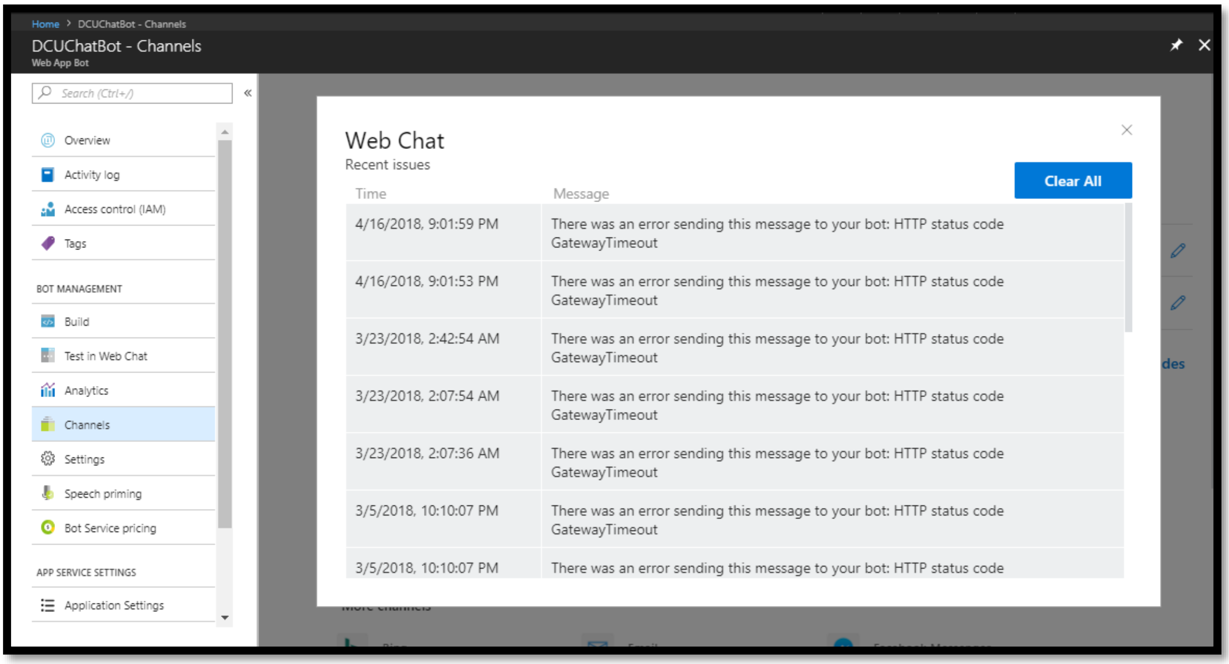
Future Work

Due to the nature of the chatbot, the number of additions that could be made to the bot are almost endless. More features can be continuously added such as weather reports, lecturer rooms, DCU library searches, and the other FAQ pages. Time can also be spent fixing any issue with the current bot, improving intelligence and dialog flow, and adding in missing feature that were in the functional spec for a sense of completeness.

As to whether the bot will continue running is another story. The bot could be extremely useful for DCU students, however keeping the bot running on Azure for 3 months has cost me almost two-hundred euro and I would not be able to keep it running any longer without removing these charges or moving the bot to another platform.

Side Note

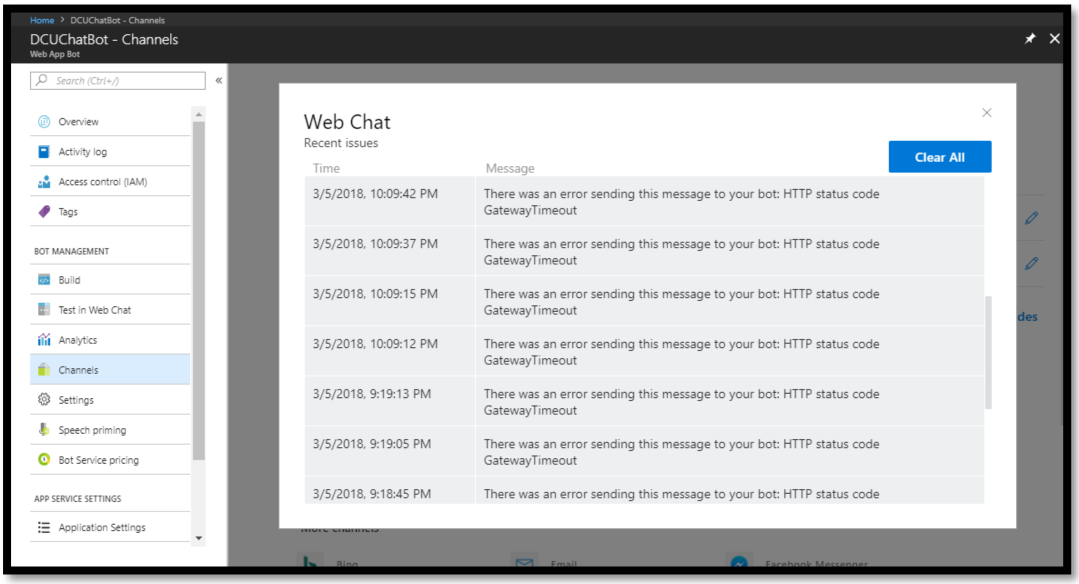
For roughly a month I was so busy publishing to azure that I forgot to push to git regularly. I have attached below some images of time stamps from Azure as proof that I was working on the project during that time. There is no publish history available on azure but I have pictures of server errors from when I published bugs, some activity logs and my dreaded bill for hosting my bot.



Web Chat

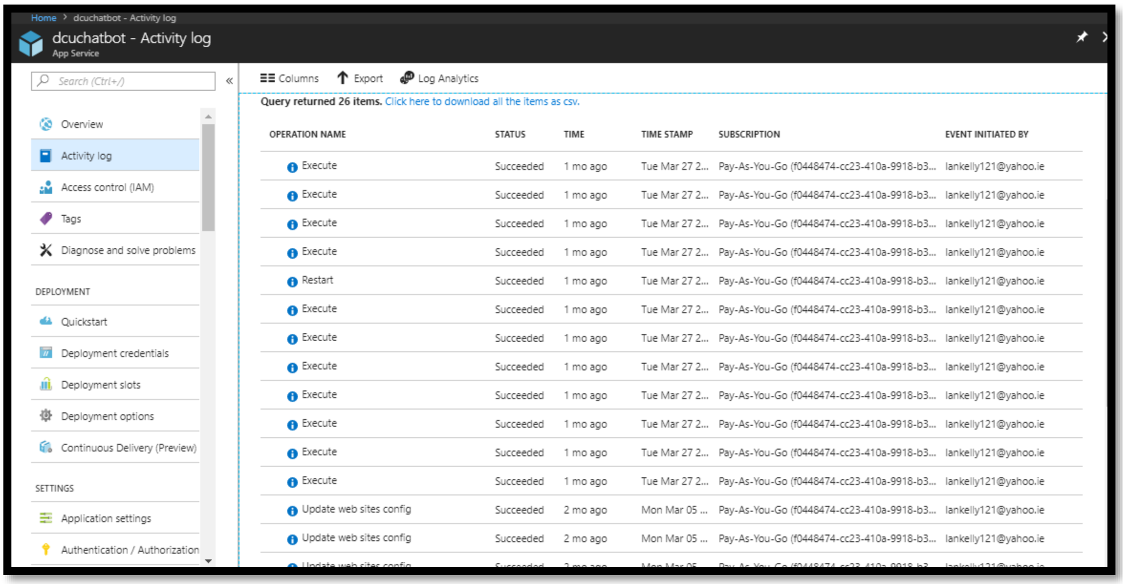
Recent issues

Time	Message
4/16/2018, 9:01:59 PM	There was an error sending this message to your bot: HTTP status code GatewayTimeout
4/16/2018, 9:01:53 PM	There was an error sending this message to your bot: HTTP status code GatewayTimeout
3/23/2018, 2:42:54 AM	There was an error sending this message to your bot: HTTP status code GatewayTimeout
3/23/2018, 2:07:54 AM	There was an error sending this message to your bot: HTTP status code GatewayTimeout
3/23/2018, 2:07:36 AM	There was an error sending this message to your bot: HTTP status code GatewayTimeout
3/5/2018, 10:10:07 PM	There was an error sending this message to your bot: HTTP status code GatewayTimeout
3/5/2018, 10:10:07 PM	There was an error sending this message to your bot: HTTP status code



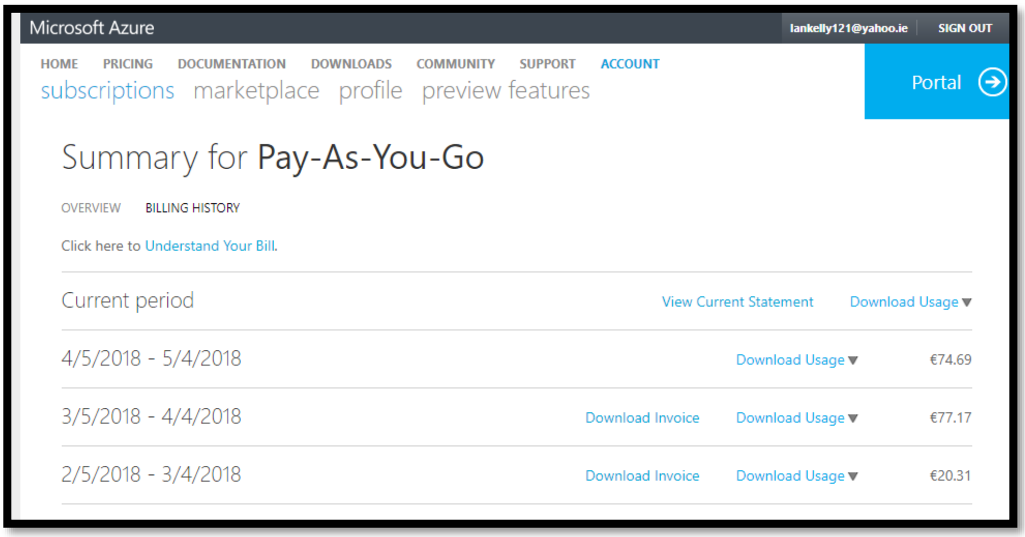
Web Chat
Recent issues

Time	Message
3/5/2018, 10:09:42 PM	There was an error sending this message to your bot: HTTP status code GatewayTimeout
3/5/2018, 10:09:37 PM	There was an error sending this message to your bot: HTTP status code GatewayTimeout
3/5/2018, 10:09:15 PM	There was an error sending this message to your bot: HTTP status code GatewayTimeout
3/5/2018, 10:09:12 PM	There was an error sending this message to your bot: HTTP status code GatewayTimeout
3/5/2018, 9:19:13 PM	There was an error sending this message to your bot: HTTP status code GatewayTimeout
3/5/2018, 9:19:05 PM	There was an error sending this message to your bot: HTTP status code GatewayTimeout
3/5/2018, 9:18:45 PM	There was an error sending this message to your bot: HTTP status code



Query returned 26 items. Click here to download all the items as csv.

OPERATION NAME	STATUS	TIME	TIME STAMP	SUBSCRIPTION	EVENT INITIATED BY
Execute	Succeeded	1 mo ago	Tue Mar 27 2...	Pay-As-You-Go (f0448474-cc23-410a-9918-b3...	lankelly121@yahoo.ie
Execute	Succeeded	1 mo ago	Tue Mar 27 2...	Pay-As-You-Go (f0448474-cc23-410a-9918-b3...	lankelly121@yahoo.ie
Execute	Succeeded	1 mo ago	Tue Mar 27 2...	Pay-As-You-Go (f0448474-cc23-410a-9918-b3...	lankelly121@yahoo.ie
Execute	Succeeded	1 mo ago	Tue Mar 27 2...	Pay-As-You-Go (f0448474-cc23-410a-9918-b3...	lankelly121@yahoo.ie
Restart	Succeeded	1 mo ago	Tue Mar 27 2...	Pay-As-You-Go (f0448474-cc23-410a-9918-b3...	lankelly121@yahoo.ie
Execute	Succeeded	1 mo ago	Tue Mar 27 2...	Pay-As-You-Go (f0448474-cc23-410a-9918-b3...	lankelly121@yahoo.ie
Execute	Succeeded	1 mo ago	Tue Mar 27 2...	Pay-As-You-Go (f0448474-cc23-410a-9918-b3...	lankelly121@yahoo.ie
Execute	Succeeded	1 mo ago	Tue Mar 27 2...	Pay-As-You-Go (f0448474-cc23-410a-9918-b3...	lankelly121@yahoo.ie
Execute	Succeeded	1 mo ago	Tue Mar 27 2...	Pay-As-You-Go (f0448474-cc23-410a-9918-b3...	lankelly121@yahoo.ie
Execute	Succeeded	1 mo ago	Tue Mar 27 2...	Pay-As-You-Go (f0448474-cc23-410a-9918-b3...	lankelly121@yahoo.ie
Execute	Succeeded	1 mo ago	Tue Mar 27 2...	Pay-As-You-Go (f0448474-cc23-410a-9918-b3...	lankelly121@yahoo.ie
Execute	Succeeded	1 mo ago	Tue Mar 27 2...	Pay-As-You-Go (f0448474-cc23-410a-9918-b3...	lankelly121@yahoo.ie
Update web sites config	Succeeded	2 mo ago	Mon Mar 05 ...	Pay-As-You-Go (f0448474-cc23-410a-9918-b3...	lankelly121@yahoo.ie
Update web sites config	Succeeded	2 mo ago	Mon Mar 05 ...	Pay-As-You-Go (f0448474-cc23-410a-9918-b3...	lankelly121@yahoo.ie



Summary for Pay-As-You-Go

OVERVIEW BILLING HISTORY

Click here to Understand Your Bill.

Current period	View Current Statement	Download Usage ▾	
4/5/2018 - 5/4/2018		Download Usage ▾	€74.69
3/5/2018 - 4/4/2018	Download Invoice	Download Usage ▾	€77.17
2/5/2018 - 3/4/2018	Download Invoice	Download Usage ▾	€20.31

External Links

Bot Hub: https://github.com/GetStoryline/awesome-bots

LUIS Docs: https://docs.microsoft.com/en-us/azure/cognitive-services/LUIS/Home

Bot Builder Docs: https://docs.microsoft.com/en-us/azure/bot-service/bot-service-overview-introduction?view=azure-bot-service-3.0

Dublin Bus API: https://data.gov.ie/dataset/real-time-passenger-information-rtpi-for-dublin-bus-bus-eireann-luas-and-irish-rail

Facebook API Changelog: https://developers.facebook.com/docs/graph-api/changelog/breaking-changes/#events-4-4

SSL Test of timetable site:
https://www.ssllabs.com/ssltest/analyze.html?d=www101.dcu.ie&latest

Dublinbusjs: https://github.com/adamisntdead/DublinBus