

**School of Science and Technology**

**CSD3999**

**Software Development Project**

**Autumn/Winter term 2017/2018**

**Date:**  20/04/2018

**Supervisor:**  Kelly Androutsopoulos

**Student Name:**  Jennifer Messer

**Student ID Number:** M00552843

**Campus:**  Hendon

**Title:** Slack Bot to Compliment SOB System

**School of Science and Technology**

Student Name: Jennifer Messer

Student Id No M00552843

Module number **CSD3999**

**I hereby confirm that the work presented here in this report and in all other associated material is wholly my own work. I confirm that the report has been submitted to TURNITIN and that the TURNITIN results are on CD attached to this report. I agree to assessment for plagiarism.**

Signature……………………………………………

Date……………………………………………………

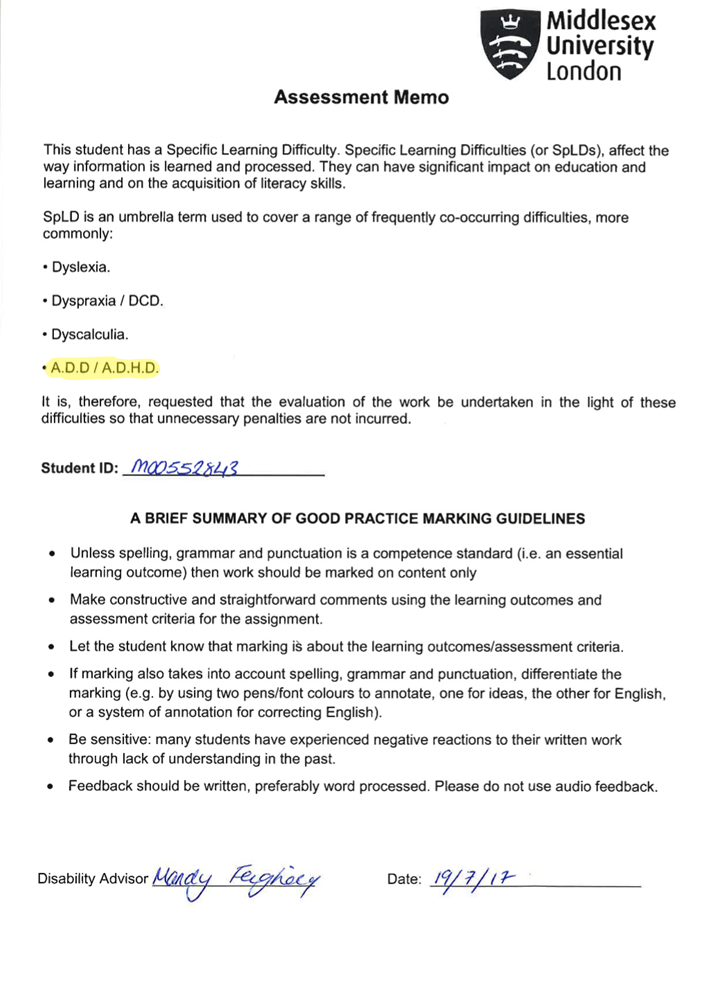


Table of Contents

[**CSD3999** 0](#_Toc512549190)

[**Software Development Project** 0](#_Toc512549191)

[**School of Science and Technology** 1](#_Toc512549192)

[Abstract 10](#_Toc512549193)

[Introduction 11](#_Toc512549194)

[Problem Definition 11](#_Toc512549195)

[Guide to report 11](#_Toc512549196)

[Background and Literature Review 12](#_Toc512549197)

[Literature review 12](#_Toc512549198)

[Educational Online Badges, Proximal Goals and Motivation 12](#_Toc512549199)

[Bots 13](#_Toc512549200)

[Slack 15](#_Toc512549201)

[Requirements specification 16](#_Toc512549202)

[Work breakdown structure 16](#_Toc512549203)

[Product review 16](#_Toc512549204)

[Stakeholders 16](#_Toc512549205)

[Functional requirements 16](#_Toc512549206)

[Quality requirements 17](#_Toc512549207)

[Use Cases 17](#_Toc512549208)

[Ask bot for report 17](#_Toc512549209)

[Ask bot for a one student’s attendance data 18](#_Toc512549210)

[Bot automatically send reminder message 18](#_Toc512549211)

[User interacts with bot for the first time 18](#_Toc512549212)

[Deliverables 19](#_Toc512549213)

[Similar Projects 19](#_Toc512549214)

[The Computer Science Student Network 19](#_Toc512549215)

[Analysis and Design 21](#_Toc512549216)

[Analysis of requirements and design 21](#_Toc512549217)

[Tools 21](#_Toc512549218)

[Database Design 22](#_Toc512549219)

[Program Design 23](#_Toc512549220)

[Sequence Diagrams 23](#_Toc512549221)

[Class Diagrams 24](#_Toc512549222)

[UML Activity Diagram 26](#_Toc512549223)

[Implementation and Testing 27](#_Toc512549224)

[Implementation 27](#_Toc512549225)

[Test plan 34](#_Toc512549226)

[Demonstration and Evaluation 35](#_Toc512549227)

[Demonstration 35](#_Toc512549228)

[Evaluation 41](#_Toc512549229)

[How well the requirements spec has been met 41](#_Toc512549230)

[Management of project and problems encountered 42](#_Toc512549231)

[Original Project Milestones 43](#_Toc512549232)

[Conclusion 44](#_Toc512549233)

[Reflection 44](#_Toc512549234)

[Lessons learned 44](#_Toc512549235)

[If the project were to be done over 44](#_Toc512549236)

[Future work 45](#_Toc512549237)

[Appendices 46](#_Toc512549238)

[References 46](#_Toc512549239)

[Guide to the materials on the USB 48](#_Toc512549240)

[My Code 48](#_Toc512549241)

[database\_test1.py 48](#_Toc512549242)

[Pybot1.py 50](#_Toc512549243)

[Adam Jarzebak’s code at time of download 53](#_Toc512549244)

[get\_user\_list.py 53](#_Toc512549245)

[sendPrivateMsg.py 53](#_Toc512549246)

[slack\_message.py 54](#_Toc512549247)

[Requirements.txt 55](#_Toc512549248)

[README.md 56](#_Toc512549249)

[.gitignore 58](#_Toc512549250)

|  |
| --- |
|  |

# Abstract

The Student Observable Behaviour System (SOB system) implemented on Middlesex Universities first year Computer Science BSc is a badge-like but distinctly unique system that tracks the knowledge of students throughout the year by staff manually checking students understand a concept and checking off the relevant SOB. Each SOB has a deadline and each member of staff has multiple students; staff must keep track of where students are at with their SOBs. This many deadlines cannot be handled efficiently by staff or students, so the idea for SOB bot was born.

The SOB bot interacts with staff and students via Slack, an online team messaging tool, and takes data from the pre-existing SOB database. The bot formats data about attendance, SOB completion and at risk students into easily digestible reports the staff can request at any time.

The bot was implemented in python 2.7.14 using the Slack web API and the database was handled with MySQL.

The project was evaluated by comparing the finished product to the requirements set out in the Requirements Specification section and using predominantly functional, though partial unit testing.

The SOB bot was proven to be good proof of concept for a Slack bot as means to interact with a large amount of student data.

# Acknowledgements

Both Franco Raimoni and Adam Jarzebak were invaluable to this project. Franco created the SOB monitor and helped get it working on the local project machine when he didn’t have time or any responsibility to help, and Adam provided ideas as to how to structure the project along with some python files that helped greatly with testing.

My supervisor Kelly Androutsopoulos has been understanding, flexible and made herself available whenever I asked for help. The students to have her as their supervisor next year are very lucky.

# Introduction

## Problem Definition

The SOB (Student Observable Behavior) system is implemented on the first year of Middlesex University’s Computer Science BSc. It is used by teaching staff and students on the course to monitor the progress students make. Staff must manually check a list of students to see which tasks (SOBs) the students have completed. Students must manually check deadlines for SOBs.

The checking process is time consuming and inefficient for staff and students. The course currently has 180 students but, as the number rises, the process will become more unsustainable for staff. Students have on average 125 SOBs, each with a deadline. Keeping on top of all of these deadlines is difficult not only for students who must not miss them, but for staff who have to enforce them. Keeping students on schedule is the most efficient strategy; staff do not have to remind students of deadlines or help students who have fallen behind.

## Guide to report

This report will cover the research, design, creation and evaluation of the project. The detailed sections are as follows:

* Background and Literature Review
* Requirements Specification
* Analysis and Design
* Implementation and Testing
* Demonstration and Evaluation
* Conclusion

# Background and Literature Review

## Literature review

### Educational Online Badges, Proximal Goals and Motivation

Badges are visual signifiers of authority, expertise, experience, and identity (Halavais, 2012). Educational badges can be offline, as with traditional scout badges, but more commonly now online badge systems are being used.

There are many non-educational online badge systems; video games and community based websites often implement badges to motivate users to continue using the service or to show other users those with badges are experienced. Badges are used as they quickly convey a message (e.g. this user is an admin, or this person is an experienced java coder) and they are easy to implement on any site as they are simply pictures. Educational badge systems benefit from students being comfortable with this type of system, the similarity also helps to gamify the work required by the course and further incentivize students (Easley and Ghosh, 2016).

Often within a badge system there will be multiple tiers, ranging from easy to acquire badges that are worth little on their own to ‘larger’ badges that require more time and possibly other badges to attain. Badge systems usually have a path a student can follow, (e.g. large badge *D* would require small badges *A,B* and *C*. Certificate *Z* would require large badges *D, E* and *F*. the path would start with *A* and end with Z.) This path however is not completely linear, students can complete tasks of the same tier at random (e.g. if a student needs small badges *A,B,C* to get large badge *D* they can complete the small badges in order *B,C,A* or *C,A,B etc*).

In the past it has been seen that external motivators negatively impact motivation (Deci, Koestner and Ryan, 1999), this should mean that badge systems are inherently flawed and they should hinder student’s learning, as motivation is very important; when low and high skill students have low motivation, they perform similarly, and poorly (*Vroom, V. H. ,1964)*.

Multiple studies have never the less shown that badge systems can work well for struggling and high passing students alike. This is because the presence of external motivators are not the only thing in badge systems that differ from the norm. ‘i*ncentives, goal assignments, need achievement, expectancies, subjective valuation of outcomes, self-efficacy expectations, and a host of other noncognitive factors have been shown to influence goal choice, intended effort, task behavior, and work performance’* (Latham and Seijts, 1999).

A badge system when implemented by an intelligent tutor can decrease avoidance behaviors (when a student is motivated primarily by a fear of doing badly) and increase interest in the subject matter, especially low-achieving and low-motivation students (Abramovich, Schunn and Higashi, 2013).

Badge systems are a type of proximal goal setting. Proximal goals are sub goals of a final (distal) goal. When a new and difficult goal is set it’s highly beneficial to put in place proximal goals as this positively affect self-referent thought, motivation, and performance (Stock and Cervone ,1990). ‘*students (in first year of college) who set and monitored proximal goals for each study session did better on the year-end examination than did those who set and monitored distal study goals for each study session.’* (Morgan ,1985).

Micro-managing learning isn’t always beneficial, when a task is novel and difficult it can be counterproductive to set a hard goal instead of a ‘do your best’ goal (Kanfer and Ackerman ,1989). This can be implemented in a badge system by making the lower levels of badges more flexible and raising the specificity of what is required to get a badge as the material gets less novel and more specialized.

One danger to consider with online badges is fakery; as of yet educational badges are rare and specialized, leading to few if any cases of people attempting to fake badges. In games however many virtual items, including badges, have been faked or fraudulently obtained. As the educational use for badges becomes more widespread the checking of badges will need to become safer (possible blockchain technology could help stealing, but not faking credentials to get badges in the first place).

### Bots

'A bot , also known as Internet bot, is a program that runs automated tasks over the Internet. Typically intended to perform simple and repetitive tasks, Internet bots are scripts and programs that enables their user to do things quickly and on a scale' (Gayer, 2018).

There are many types of bots, they can be loosely categorized into ‘good’ bots and ‘bad’ or ‘malicious’ bots. Good bots have official monikers as they are legal, which makes it easier to discuss them individually. Some common types of good bot are:

|  |  |
| --- | --- |
| Bot | What it does |
| Monitoring | Bots that monitor website availability and the proper functioning of various online features. |
| commercial crawlers | Spiders used for authorized data extractions, usually on behalf of digital marketing tools. |
| search engine | Bots that collect information for search engine algorithms, which is then used to make ranking decisions. |
| feed fetcher | Bots that ferry website content to mobile and web applications, which they then display to users. |

Only the worst bad bots get to be named: nitol, cyclone, sentry MBA. Some common types of bad bots are:

|  |  |
| --- | --- |
| Bot | What it does |
| impersonators | bots that assume false identities to bypass security solutions.  Commonly used for DDoS attacks |
| Scrapers | Used for unauthorized data extraction and reverse engineering of pricing methods |
| Spammers | Inject spam links into forums, discussions and comment sections |
| hacker tools | scavengers that look for sites with vulnerabilities to exploit for data theft, malware injection etc |

(Ziefman, 2017)

A chatbot is a machine conversation system which interacts with human users via natural conversational language (Shawar and Atwell, 2005). Machine learning and other AI techniques can be used to analyze language used by humans and attempt to recreate a human conversational experience, this results in a more natural use of language than other methods but it has big drawbacks. A chatbot trained on human speech will talk like a human, and humans often say rude things. Microsoft’s AI chatbot ‘Tay’ was taught to say racist things within less than a day of it being open to the public on twitter to interact with (Wakefield, 2018).

The average chatbot searches for key words and phrases in the user’s message like ‘weather’ or ‘bank balance’ and then sends the reply associated with that word. This system is flawed as you can tell a bot ‘don’t tell me the weather’ and it will immediately tell you the weather simply because you used a trigger word; these simple bots often do not understand context. The best a basic chatbot can do with context is hold the user’s previous few messages and regurgitate their name or another piece of information later. This doesn’t mean basic chatbots aren’t useful; chatbots are often used by businesses as an alternative to having an automated phone attendant that speaks to the user over the phone. This is cheaper for the business and the customer, and is easier for the customer if they dislike phone calls or find conversing over social media more comfortable.

### Slack

Slack is a cloud-based set of proprietary team collaboration tools and services, founded by Stewart Butterfield. Slack’s features include: persistent chat rooms (channels) organized by topic, private groups, direct messaging, fully searchable chat space including files, conversations, and people, integrations with a large number of third-party services including Google Drive, Trello, Dropbox, Box, Heroku, IBM Bluemix, Crashlytics, GitHub, Runscope and Zendesk, community-built integrations, app directory consisting of over 150 integrations that users can install. (En.wikipedia.org, 2018).

Slack has a feature where you can create and implement a custom chatbot. In Slack bot users can post messages, react to users, monitor channel activity, upload files, be invited to and kicked out of channels and make interactive buttons. Bot users are controlled *‘programmatically via a bot user token that accesses one or more of Slack's APIs’* (api.slack.com, 2017).

# Requirements specification

## Work breakdown structure

## Product review

The aim is to develop a slack bot that interacts with the existing Student Observable Behavior (SOB) system currently implemented on the first year of the Computer Science BSc at Middlesex University.

## Stakeholders

* Staff teaching on the first year of Computer Science BSc
* Students on the first year of Computer Science BSc
* Faculty
* University administrators

## Functional requirements

* User must be able to initiate a bot interaction
  + Private messaging the bot
  + Tagging the bot or using a command in a group conversation
* Bot initiated interaction
  + Reminders and reports sent to private or group chats
* Allow remote client connections via the internet
* access SOB database
* send appropriate error messages to the user/client
* create and deliver reports to individual teachers
* report must contain
  + list of students who are behind
    - with list of overdue sobs
  + list of student with bad attendance
    - with dates they have/have not attended
  + table showing all students and their current number of sobs

## Quality requirements

* Bot must be:
  + Server application implemented in Python 2.7.14
    - using the slack API
  + Accessible from a multitude of devices via Slack including
    - Desktop Slack app
    - Slack in browser
    - Slack mobile phone/tablet app
  + Simple to use
  + Fast
    - The data must be returned after a request within 1 minute
  + Use friendly/pleasant language
  + Easily maintainable
  + Accessable
* Student data must be secure
* User guide must be easy to understand and clearly presented

## Use Cases

### Ask bot for report

|  |  |
| --- | --- |
| Name | User message bot asking for a general report |
| Identifier | UC01 |
| Version | V1.0 |
| Initiator | User/teacher |
| Goal | Retrieve data and format into a full report given to user |
| Assumptions | User has already been put in contact with slack bot |

Main Scenario:

1. User sends a private message to the bot.
2. A word or phrase in the user’s message triggers the *report* function
3. the bot retrieves data about the user’s students
4. The bot formats the data into an easy to read layout
5. The bot posts the report as a private message to the user

### Ask bot for a one student’s attendance data

|  |  |
| --- | --- |
| Name | Specific data retrieval |
| Identifier | UC02 |
| Version | V1.0 |
| Initiator | User/teacher |
| Goal | Retrieve data and format into a message given to user |
| Assumptions | User has already been put in contact with slack bot |

Main Scenario:

1. User sends a private message to the bot.
2. A word or phrase in the user’s message triggers the *getStudent.attendance* function
3. the bot retrieves data about the student’s attendance
4. The bot formats the data into an easy to read layout
5. The bot posts the report as a private message to the user

### Bot automatically send reminder message

|  |  |
| --- | --- |
| Name | Auto reminder |
| Identifier | UC03 |
| Version | V1.0 |
| Initiator | Bot |
| Goal | Remind all users of upcoming deadlines |
| Assumptions | User has already been put in contact with slack bot |

Main Scenario:

1. A SOB deadline saved in the database is a few days away
2. The bot gets the data about the deadline (date, time, which students have already completed the work)
3. The bot sends a reminder to all students who have not completed sob
4. The bot sends a reminder to all teachers with students who have not completed the SOB with a list of all the students who have not completed the SOB.

### User interacts with bot for the first time

|  |  |
| --- | --- |
| Name | Intro to slack bot |
| Identifier | UC04 |
| Version | V1.0 |
| Initiator | User |
| Goal | Make sure user knows how to properly utilize bot |
| Assumptions | User has slack account and has been added to a workspace where the bot has been installed |

Main Scenario:

1. The user sends a private message to the bot
2. The bot presents the user with 2 buttons, student and staff
3. If the user selects staff the bot asks the user for their employee number
4. If the user selects student the bot asks the user for their student number
5. The bot searches the database for the correct entry pertaining to the user
6. The bot puts the entry about the user into a message and shows two more buttons, yes and no
7. The bot asks the user to confirm that the entry matches their identity
8. If the user selects no repeat from step 2
9. If the user selects yes the bot sends the user guide to the user

### Deliverables

* Python and SQL code
* User guide in plain text
* Final report detailing bot

## Similar Projects

### The Computer Science Student Network

The Computer Science Student Network (CS2N) is a badge system for the Computer Science and Robotics programs on the Carnegie Mellon CS-STEM Network ([http://www.cs2n.org](http://www.cs2n.org/)).

Each student has a profile page where they access their badges and potential badges they are on a path to learning. Badges are earnt by the student proving they have the required skills. Proof of skill (and therefore attainment of badges) is achieved multiple ways, outlined in *Figure. 1,* including:

* Activities – in class tasks
* Artifacts – file submitted by student, proof of work (e.g. math working)
* Quizzes – online test
* Endorsements – instructor signing off on a piece of work

A specific collection of smaller badges is needed to get a larger one, badge sizes shown in *Figure 2.* A path from small badges to industry badge is laid out, but within levels there is not always a specific order the badges need to be completed in (i.e. not all small badges in a path must be completed in order but they must all be completed to gain a specific medium badge). The badge is then awarded in Carnegie Mellon’s name by the instructor that was leading the class.

each badge is recorded and can be shown as an attainment in its own right; the external certification however is the only thing normally recognised as a qualification.

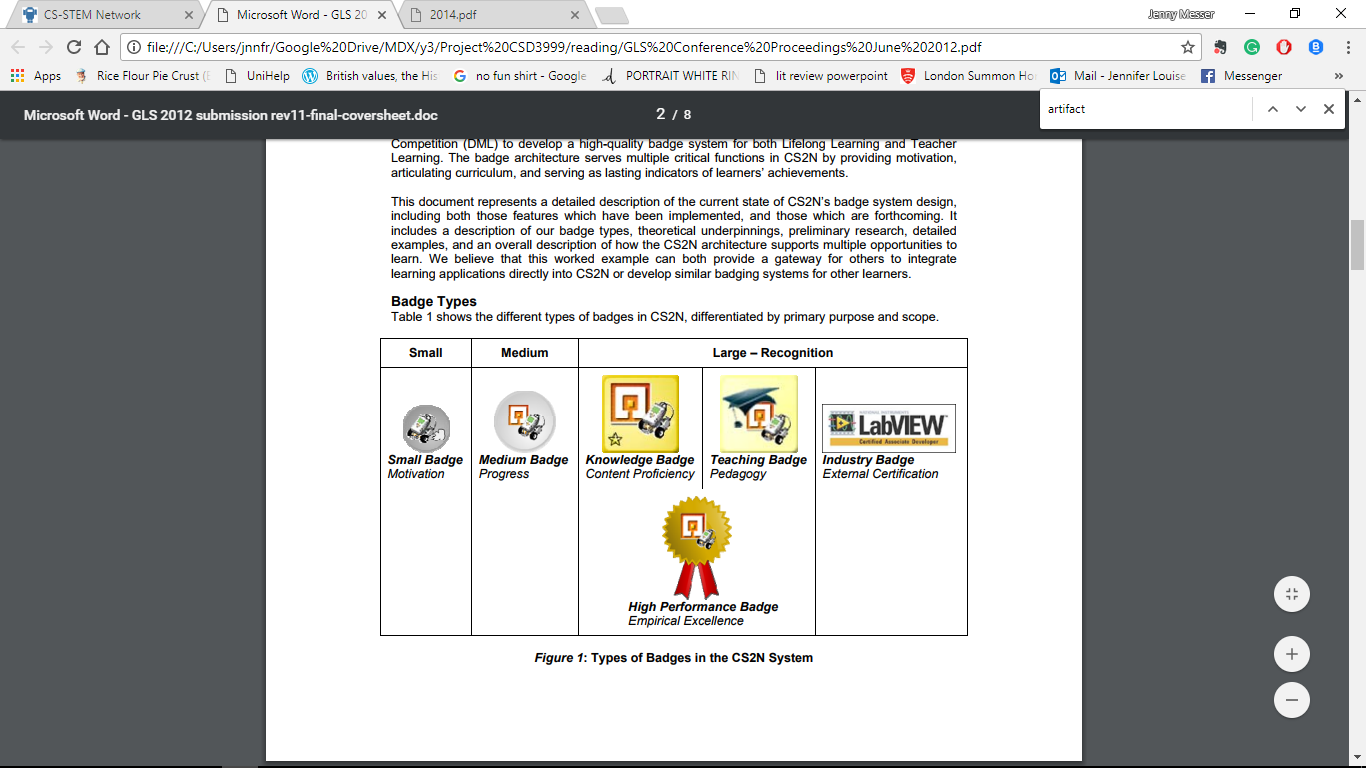


Figure 2 - levels of badge attainment

Figure 1 – badge level correspondence chart

|  |  |  |
| --- | --- | --- |
| Level - 1 small and medium | Level 2 – knowledge/teaching/performance | Level 3 – external certification |
| **Activity badges**  Earned with:   * classroom activities | **skill badges**  earned with:   * Activities * artefacts * quizzes * endorsements * evidence | **Certificates**  Earned with:   * Badges * Artefacts * Endorsements * final exam |

# Analysis and Design

## Analysis of requirements and design

### Tools

Some of the major design decisions taken were what tools to use. The tools used are:

#### Slack

Slack was chosen as the medium of communication between the user and the program as firstly the users already use Slack. Integrating a slack bot into already existing channels would be a very quick and seamless way to get users used to using the new system. Secondly Slack allows for bots the users can talk with to be easily created, which facilitates communication between program and user. Thirdly, Slack comes with multiple APIs available to use for free. The API used in this project is the web API. The web API allows the program that controls the bot to be kept on a private server and edited whenever it is convenient.

When the bot is placed into a workspace users can initiate interactions with the bot at any time and the bot can send messages/reports to the users at specific times. Slack has channels in workspaces and private chats between users, so the bot can message multiple people at once in a group space or privately message users. This means users only get data they need or ask for and aren’t inundated with messages containing data other people need.

#### Python

Python was chosen as the main language to write the code in as Python is a versatile high level language with thousands of third party modules available to use. (python.org, 2018) Also I have experience coding in Python. Python also has pip, a built in package management system that makes it easy to download and install packages in Python system wide or in a virtual environment.

#### Virtualenv

Virtualenv is a tool to create isolated Python environments (virtualenv.pypa.io, 2018). A virtual environment was used so the system could be tested without any other packages not needed for the system, altering the result.

#### XAMPP (Apache and MySQL)

XAMPP is a free and open source cross-platform web server stack package that contains Apache HTTP server and MySQL data management system (SourceForge, 2018). Apache was used to host a local version of the SOB website, used primarily for testing and creating fake data to go into the database. The MySQL server was used for the database containing all the data on student, teachers and SOBs needed for the system. MySQL was chosen as it’s easy to work with in python; with a few lines of code any piece of data can be accessed from the database, allowing for quick retrieval of data about SOBs and students from multiple tables.

XAMPP was used as the bundle of servers was convenient and it is easy to setup and use. The user interface is clear, press start next to either ‘Apache’ or ‘XAMPP’ to start the corresponding server.

#### Flask

Flask is a micro web framework written in Python and based on the Werkzeug toolkit and Jinja2 template engine (Flask.pocoo.org, 2018). Flask is used in the project to host the app locally.

Flask was chosen as, being only a micro-framework, it is very lightweight. Applications built with Flask have little dependency as there are little to no libraries to be dependant on updates for. This means the app is more secure.

#### ngrok

Ngrok exposes local servers behind NATs and firewalls to the public internet over secure tunnels (Ngrok.com, 2018). Ngrok was used with Flask to allow the app to run locally but be accessible by slack via the internet.

When an event happens in the Slack workspace where the bot has been deployed, Slack sends an HTTP POST request to a url. Ngrok provides this URL and allows a two-way communication between Slack and the app.

### Database Design

The database structure is the same as that of the SOB database. If the slack bot was to be implemented alongside the SOB system they would both draw data from the same pre-existing database, so it makes sense for the slack bot to be made with that pre-existing database in mind. This makes integrating the system into the current system more seamless.

### Program Design

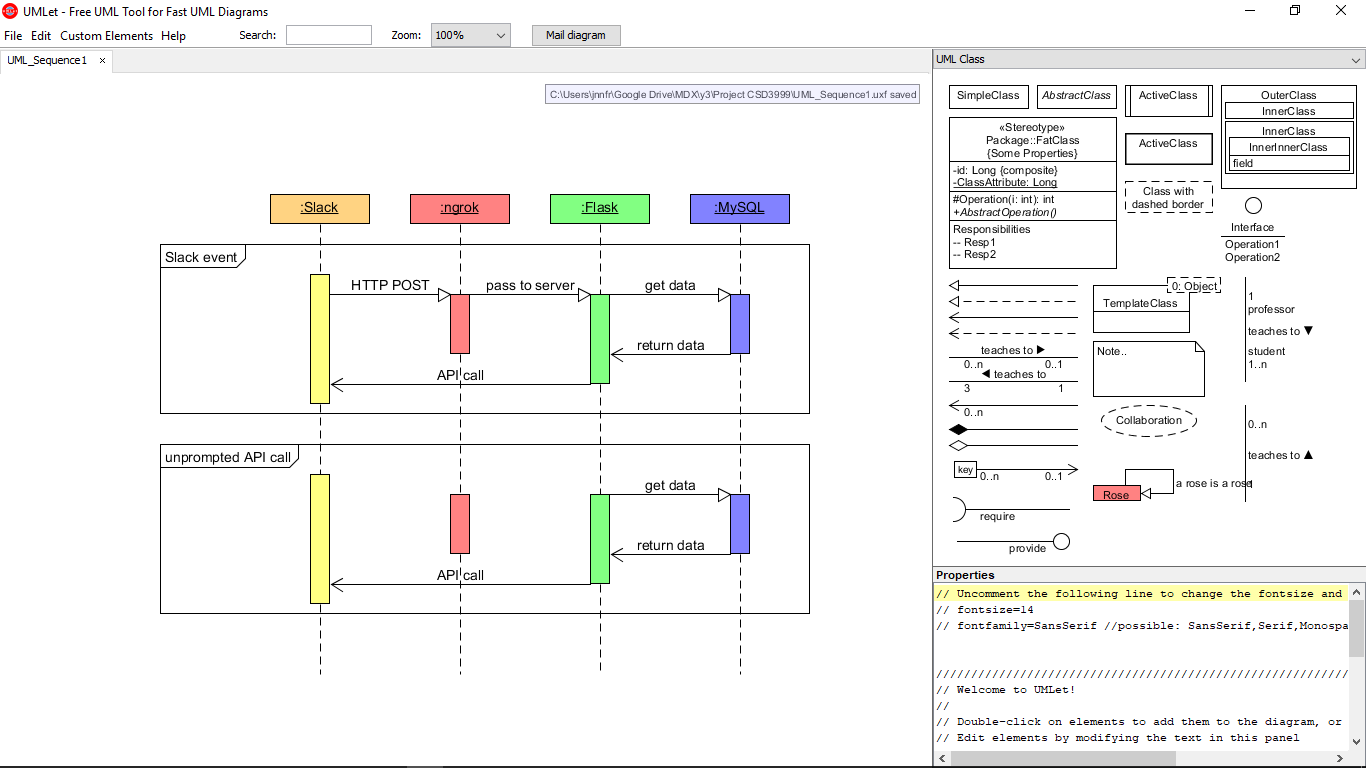
There are many options as to how the user could communicate with the app through slack. Slack provides things such as drop down menus and clickable buttons; for this project only text based communication is used. The Slack API available for python is the web API, only plain text communication is supported with this API. The real time messaging API provided by Slack would have been useful but it is not supported with Python, the language used in the project.

Because only the web API could be used the program contains a lot of string handling. This inadvertently makes the program more accessible; anyone who knows the commands can type or use speech to text to enter commands into Slack. The system is accessible for these reasons:

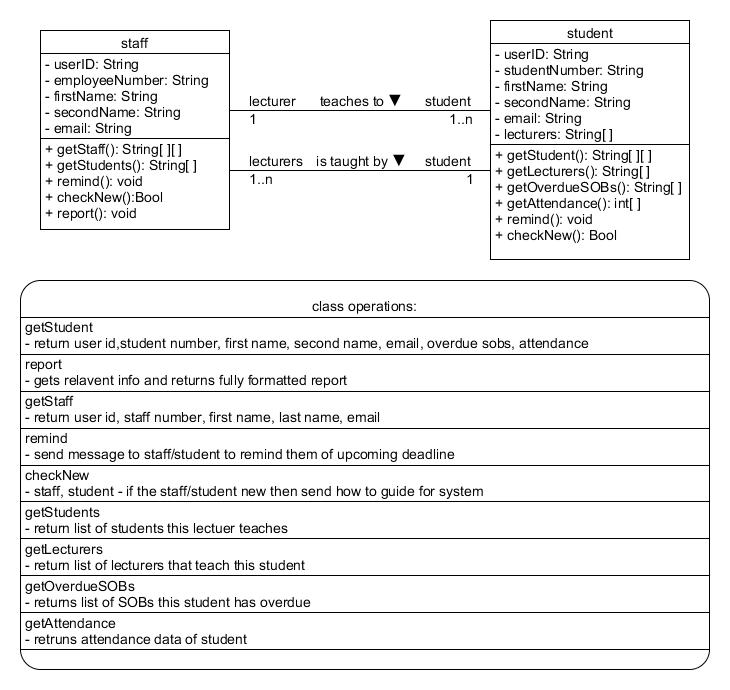
* Text based systems are very accessible as text can be converted into large print, braille, speech, symbols or simpler language
* it’s a simple layout
* all functionality can be accessed from a keyboard
* the program is not time sensitive and can be accessed at any time so the user can take their time if they need it (World Wide Web Consortium, 2008)

The program handles an HTTP POST request by checking for key words in the string and using the key words to determine what needs to be done with the rest of the string.

Sequence Diagrams

The python program, which is the brain of the app, runs on the Flask server. When an event happens in Slack an answer is always returned from the program. Sometimes, for example with the automated weekly report, the program will send something to Slack without being prompted first. 

## Class Diagrams

Due to time constraints the prototype did not get all data on students/staff and put them into an object to be accessed later, as was the original plan. The following diagram describes what those objects would have looked like. The prototype program gets specific data needed for a proof of concept operation, the report() function in the staff class below. 

## UML Activity Diagram

# Implementation and Testing

## Implementation

The first step to starting the project was to get a local version of the SOB monitor working. The SOB monitor is available on bitbucket - bitbucket.org/mdxmase/sobmonitor/src (Raimondi F, 2016). The repository was downloaded and an Apache server was set up using XAMPP. XAMPP was already downloaded on the machine used to create the program so it was easy to get the server up. Getting the login screen to appear in the browser was straight forward, just place the SOB monitor files in C:\xampp\htdocs. To navigate to the other pages first the user must log in, but without a database there is no username and password that will let the user passed the initial page.

The next step was to set up the database with MySQL. I have never used MySQL or PHP before so figuring out how to get things set up properly was difficult. Eventually it was discovered that phpMyAdmin could be accessed via the browser and be used to upload the db\_setup.sql file that came with the SOB monitor. The database structure was set, but the database was empty. phpMyAdmin allows for manual entry of data so one admin user was entered into the proper database; unfortunately, it still didn’t work. Attempts at debugging were fruitless and lead to a big setback in the schedule.

The creator of the SOB monitor, Franco Raimondi, looked over the files on the primary project machine and pointed out the config.php files in C:\xampp\htdocs\m.student and C:\xampp\htdocs\admin didn’t have the correct connection information to connect to the new empty server. After changing this the website functioned properly.

Now a login is possible the SOB monitor site was studied. To properly understand what functionality the users may want from the Slack bot first the already existing functionality must be understood. To see the site in action, and to later test with, fake data was put into the database. Data was added firstly using the SOB monitor’s ‘upload student data’ feature, where an excel spreadsheet containing student data can be automatically uploaded and stored in the database, and secondly using the manual input of data one SOB at a time using the site’s SOB creation tool.

Adam Jarzebak, another final year Computer Science student at Middlesex University, was working on a similar project in his spare time before becoming busy with coursework. He provided his code, available on github at <https://github.com/jarzab3/py-sobbot>, in the appendix and on the USB. At time of download the git consisted of:

* three python files
  + get\_use\_list.py
  + sendPrivateMsg.py
  + slack\_message.py
* README.md
* requirements.txt,
* .gitignore
* sampleApiResponse.txt
* sobMonitor\_database.sql

Adam’s code needed to be tested on the local machine to quickly prove a connection could be made between Slack and the machine. In the README basic directions were given on how to set up the system, but full instructions had not yet been written. Per the README, python2 was installed in a virtual environment using pip. The next task was to install psql, but as MySQL was used instead of PostgreSQL, the MySQL database connector for python was downloaded manually instead. Finally using pip the requirements.txt file was used to install multiple python packages in the virtual environment.

Slack\_message.py was tested; the file was run and a message appeared in Adam’s previously set up Slack workspace. From here on there was no guidance from Adam or his work.

Only the user who created the bot in the workspace has control over it, so a new bot and workspace was created. Firstly, a new workspace was created by going to slack.com/create and filling in the relevant information. The outgoing webhooks integration must be added to the workspace so the app can send HTTP POST requests to the system. This can be done by searching the Slack app directory custom integrations for outgoing webhooks, and selecting the *add outgoing webhooks integration* button. This provides a token that will be sent with the outgoing payload and can be used to verify the post came from the correct Slack workspace. The webhook needs a URL to post to, this is set up later.

Slack bots work through Slack apps, so a Slack app had to be created. This can be done by going to api.slack.com/slack-apps, selecting create app, filling in the name of the app and selecting the workspace it should be implemented in. Once the app has been created and added to the correct workspace a bot user can be added by selecting the *bot users* tab and press the *add bot user* button.

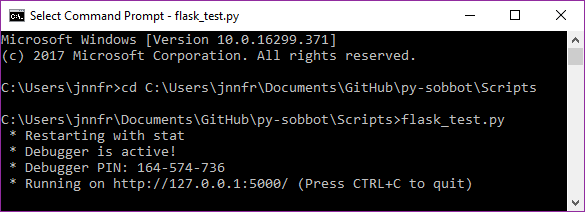
The next step was to get the authorization codes generated by Slack, so when messages are passed to the program it can be sure the messages are coming from the right app. This keeps the system secure. The required codes are:

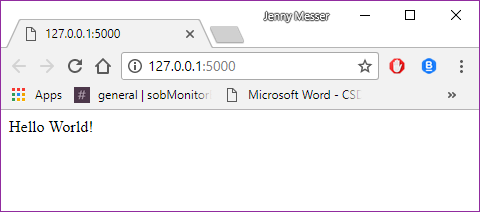
* bot verification token
* slack verification token
* slack webhook token

As previously mentioned the webhook integration needs a URL to post to, this is where ngrok and flask come in. The python program can run on the local Flask server and then ngrok exposes the server to the internet and provides a temporary URL.

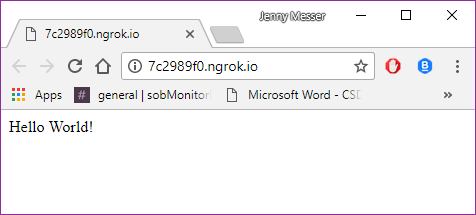
Flask was first downloaded and installed in the virtual environment using pip. A test python file *flask\_test.py* was created to see if Flask was running properly.

The test program was run using the command prompt so the port the server was running on could be checked. Once the server was up a <http://127.0.0.1:5000/> was accessed via the browser and the text ‘Hello World!’ was shown, indicating success.

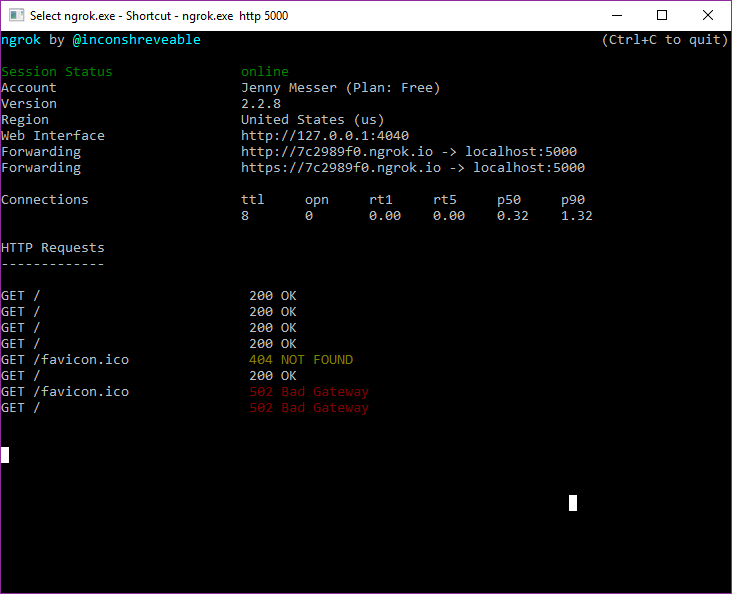


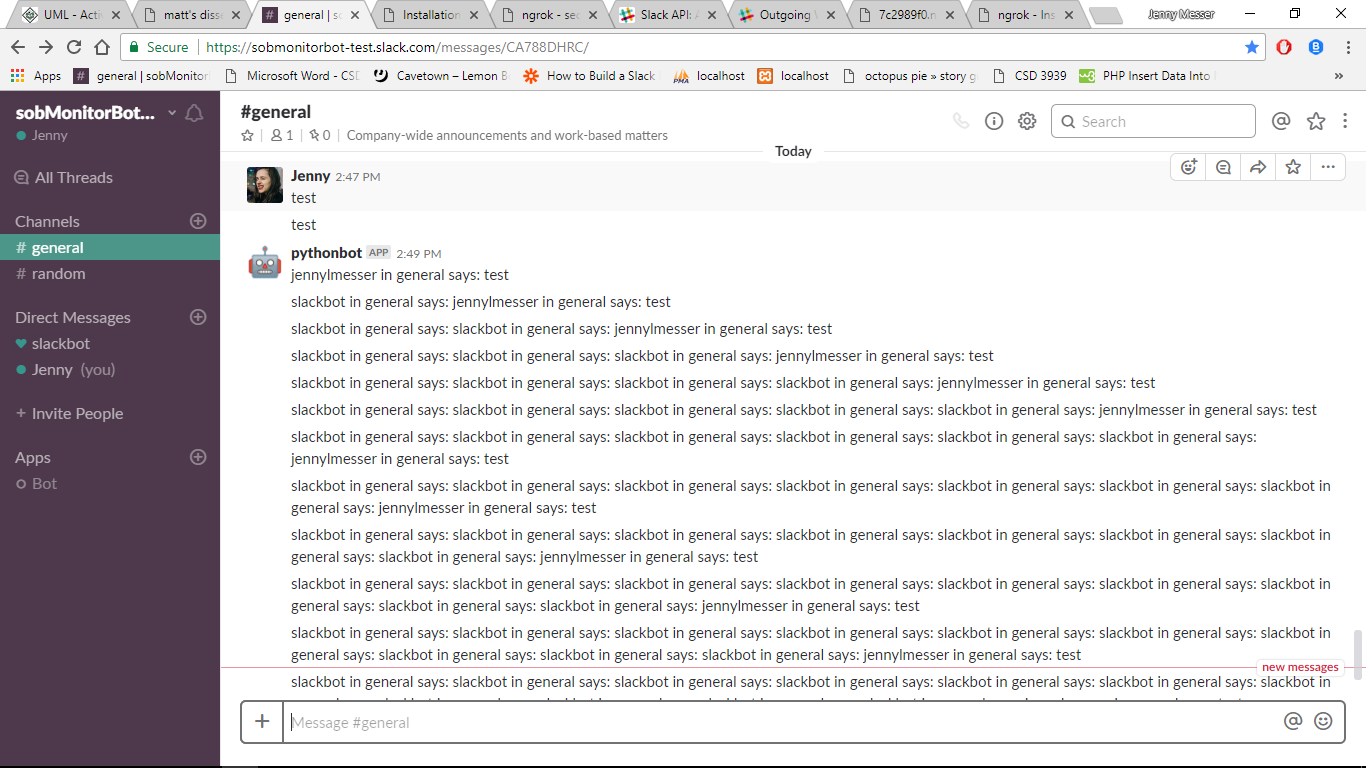


Next ngrok was downloaded and started using the ngrok.exe file. Ngrok asks what port it should run on, it must be run on the same port as the Flask server. In this case it was port 5000. To test if the tunnelling works a similar test to the Flask test can be done. The test program is run, but instead of accessing <http://127.0.0.1:5000/>, the ngrok URL provided (<http://7c2989f0.ngrok.io/>) was accessed, again displaying the ‘Hello World!’ text, showing the tunnelling worked.



The ngrok command line interface also shows information about connections made and if the connection was successful. This is where you get the ngrok URL from.



At this point all the tools are in place for a two way messaging between Slack, the program and the database. A test to see if a message could be sent from slack and a response returned is undertaken. When a user posted in Slack the bot was supposed to return the users name, the channel they posted in and what they said, just to show all that data had been received and re-sent correctly. This created in infinite loop which had to be stopped manually, as the bot recorded itself posting in Slack and then reposted what it said over and over. 



As a test to show if data could be passed between the project and Slack it was very successful. A problem was found however, when the program is running no print statements or error statements show up in the python shell. A lot of time was spent on experimentation, trying to find out where the error was. In fact there was no error, the program ran fine, just Flask needs it’s own debugging tool to be used. Again after this was found out, more time was spent trying to understand the Flask debugger. As the deadline was approaching attempts to understand the debugger were abandoned and instead print to text file statements were used to identify errors. This method was not very efficient; debugging was slow and the actual error that occurred could not be shown, only at what line the error occurred.

Once the text file based debugging was implemented the final program could be started. The first step was to access the MySQL database. Database\_test1.py was written and was functional immediately. This program didn’t include Flask so the results could be printed to the console.

The final program pybot1.py was created. The first step was to integrate the parts of the previous programs that worked, i.e. sending messages, receiving messages and accessing the database. The function getAllStudents() was created to test that the program could receive a specific request, get data from the database, format it and return it to Slack. The function inbound() is triggered when an HTTP post request is sent to the server. inbound() primarily consists of a series of if/else statements to determine what is to be done with the data. For instance getAllStudents() is called from inbound() when the plaintext in the request is equal to ‘get list of all students’.

Once this was functional something meaningful needed to be done with the data, namely the creation of a student report.

Three new functions were created. studentReport(searchBy,searchText) creates a report on student with name *searchText.* SearchBy refers to the method of retrieving the data, by name or by student number. This function searches the student table for a matching name/student number and then saves the student’s ID to a variable. That ID is then used to identify entries in other tables in the database that relate to that student. The report function returns the students name, email address, student number, and the SOBs they have completed. Send\_message(channel\_id, message) sends the message *message* to the channel with ID *channel\_id.* The send message code was in the program before the creation of the function, but this put that code into a reusable function. Test() sends the text ‘Hello World!’ when an HTTP GET request is sent to the server.

Due to time constraints the report function was not fully implemented, and none of the other intended functions were added to the program. The program still works as a good proof-of-concept prototype.

## Test plan

The testing will be primarily functional, with some aspects of unit testing. Unit testing will show that the program code works correctly, and the function testing will show the program fulfills any requirements initially set out for the system. (Hughes and Cotterell, 2011). The testing will be done by pasting the code to be tested into a new file and adding other code to test it’s function.

The unit testing will look at specific chunks of code and see if they do what they’re supposed to do. Each function in the final program will be individually tested, apart from studentReport() and inbound() which will be broken down into smaller pieces.

studentReport() will be broken down into: strip string correctly, using student name to find entry in table, using student number to find entry in table, getting the SOBs related to the student and getting the names of the SOBs.

Inbound() will be broken down into: does an HTTP POST request activate the function, get plaintext, do right thing with plaintext.

Code outside functions that will be tested is if the program connects to the database properly, if the verification tokens are correct and if the flask server runs correctly.

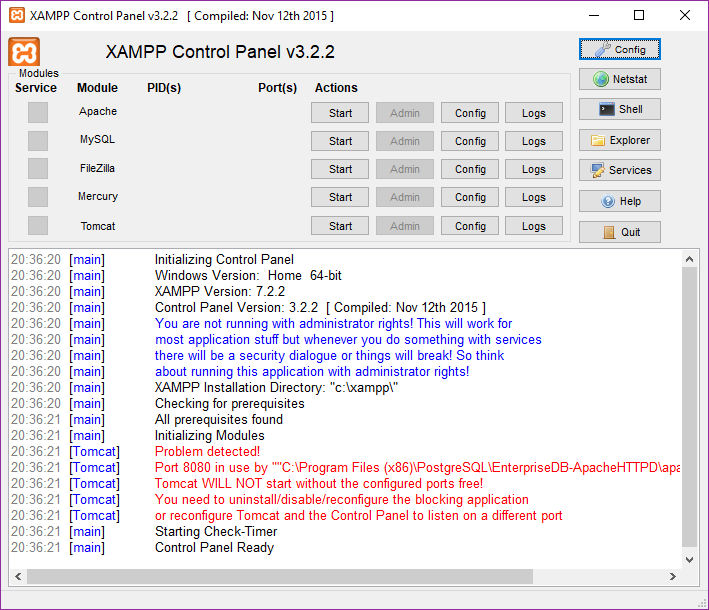
The functional testing will focus on the requirements specified earlier in the report. This will include:

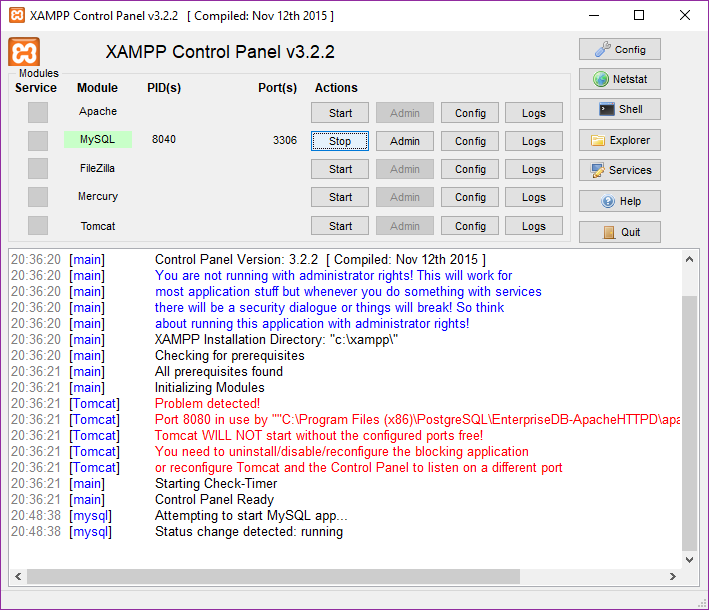
* Getting student reports
  + for different students
  + using student name
  + using student number
  + for students with no observed sobs
  + for students that don’t exist
* testing if commands work with extra spaces and characters
* testing if commands work with differing amounts of upper and lower-case letters
* if the different users sending the commands effects the result

Some requirements were not attempted so these will be mentioned in the evaluation, not the testing section.

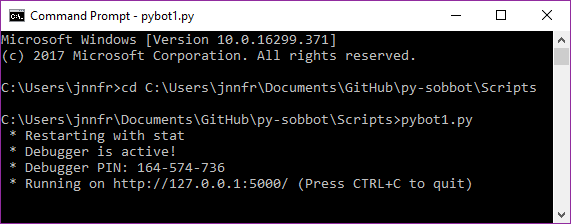
# Demonstration and Evaluation

## Demonstration

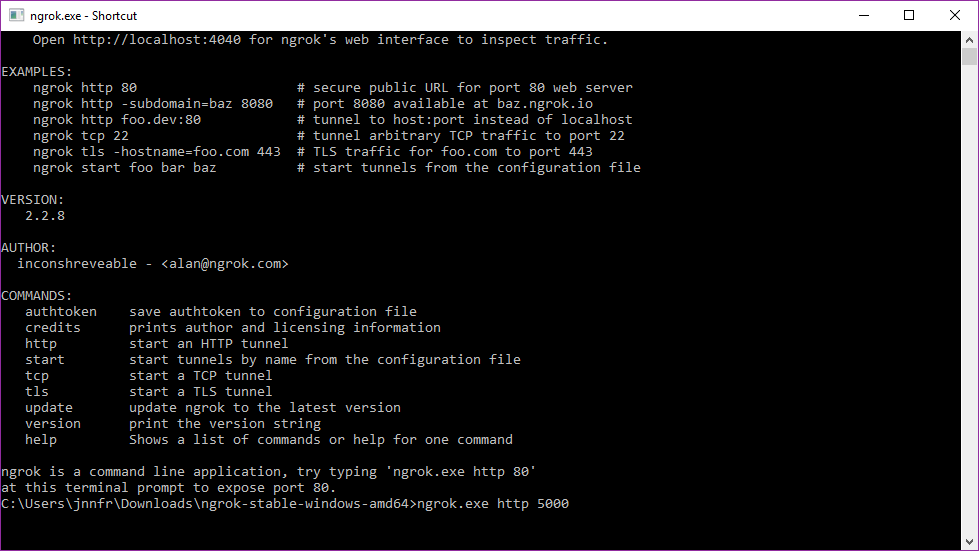
Start the XAMPP control panel. Only MySQL is needed for the program to run, so as long as there is no error for MySQL and the final ‘Control Panel Ready’ message is present. 

Press the start button next to MySQL to start the server. Once the running confirmation message appears move on to the next step.

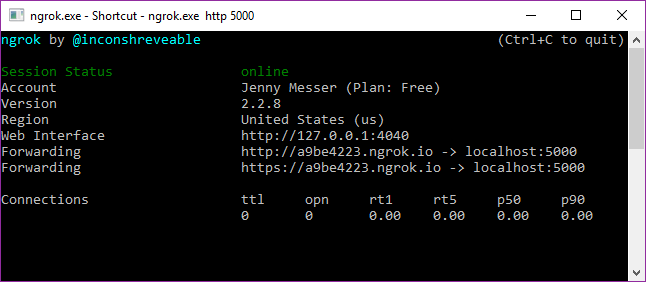
Start the program either by double clicking the pybot1.py file, running it in the python IDE or using the command line. Here the command line is used as this way we can check what port the Flask server is going to run on.

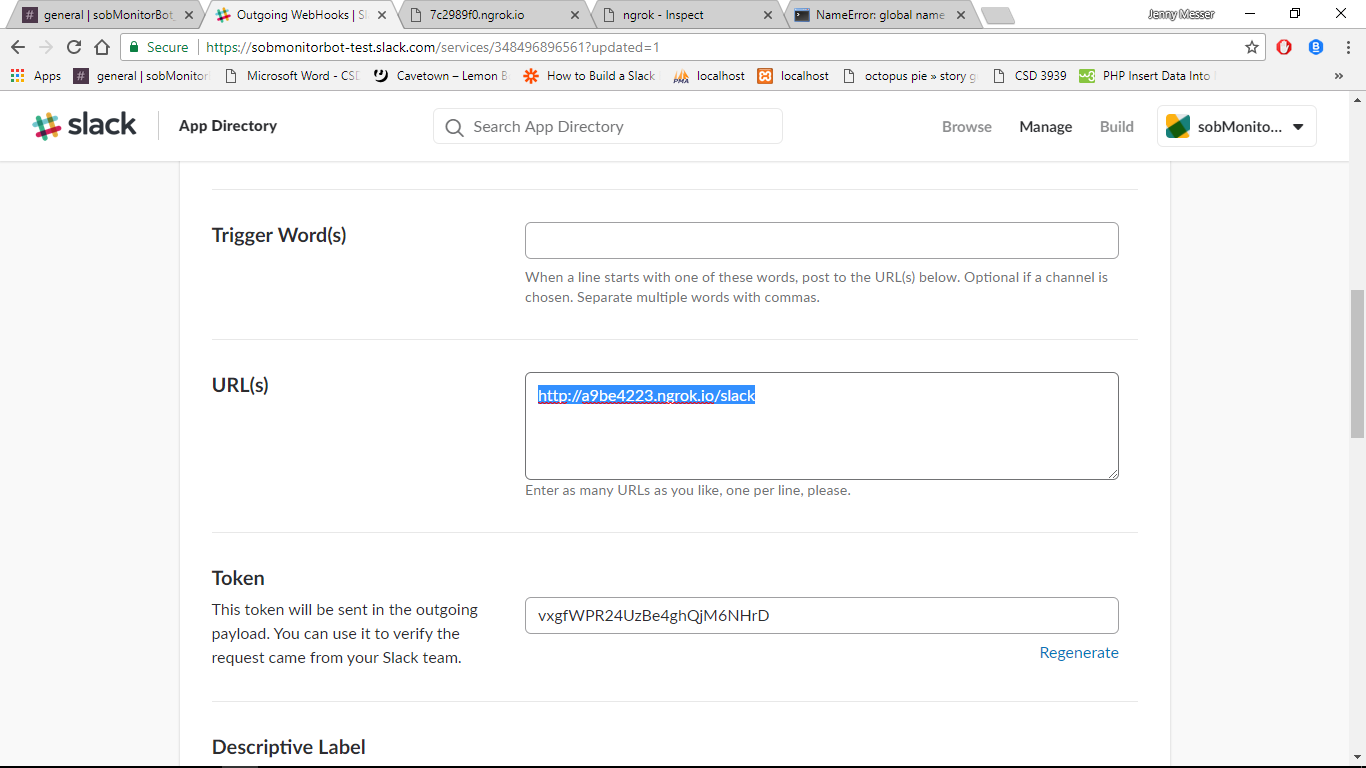


The server is running on port 5000.

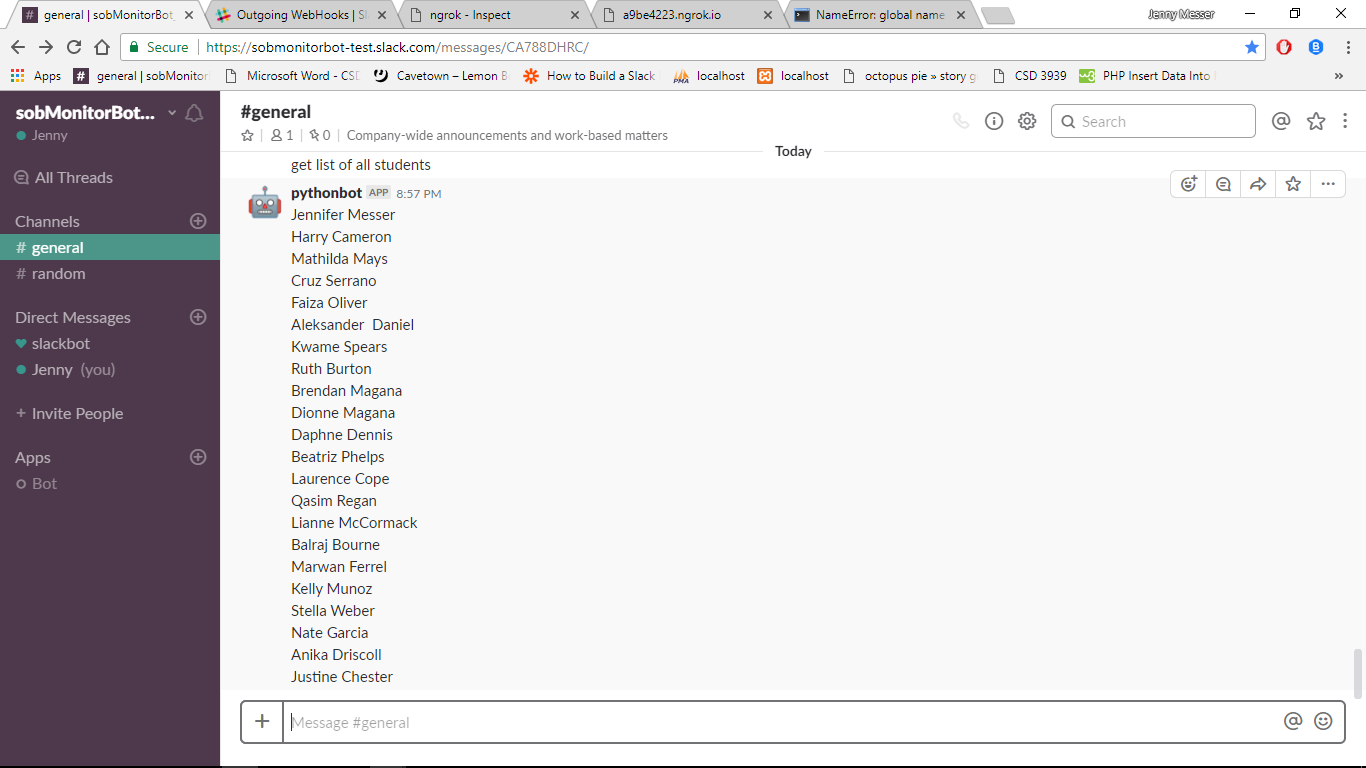
Start ngrok by double clicking ngrok.exe or accessing it via the command line. Once ngrok has been started this terminal will appear. Enter the port the Flask server should appear on, so enter *ngrok.exe http 5000*

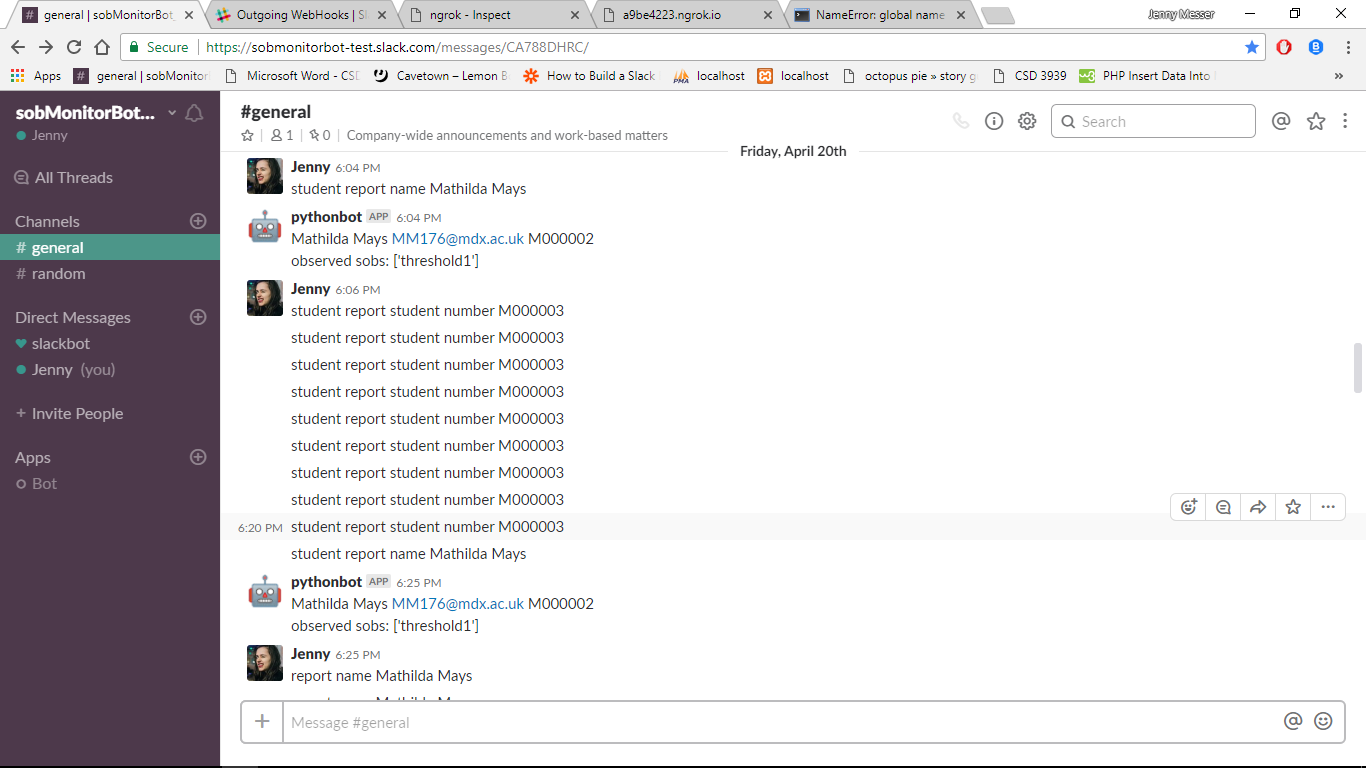
After executing this command this new interface will appear showing the new ngrok URL.



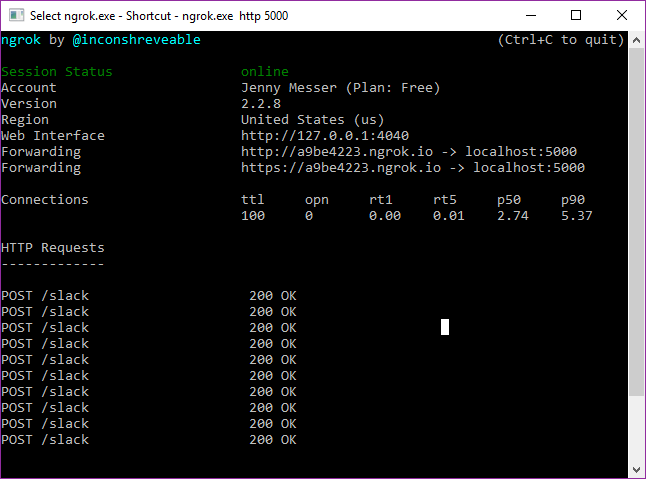
The new ngrok URL must be entered into Slack in the outgoing webhooks page. Enter the ngrok URL and add \slack to the end. When Slack sends an HTTP POST request to the program, the program will know it’s from Slack. 

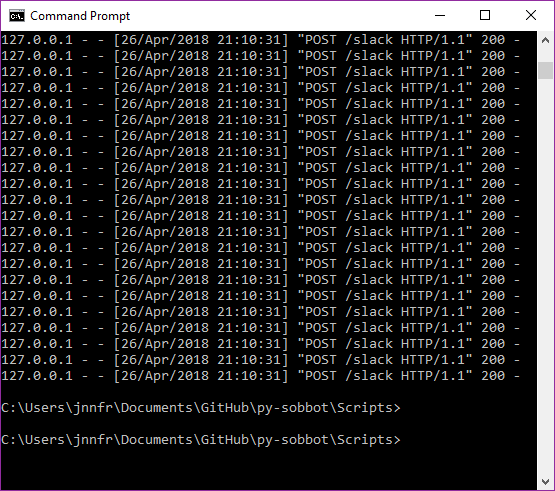
Now we can go to Slack, log in and navigate to #general. This is where the commands must be posted. Two commands will be shown: report and get all students. The way the command is worded is key to getting it to work. To get the list of all students the command must read exactly ‘get list of all students’. To get a report on a student the command must read ‘student report’ then either ‘name’ or ‘student number’, then finally the full name of the student.





To check actual POST requests are being sent, check the command prompt or ngrok panel

. 



## Evaluation

### How well the requirements spec has been met

|  |  |
| --- | --- |
| Requirement | How well it was met |
| User must be able to initiate a bot interaction | As proved in the screen shots, this works. |
| Private messaging the bot | This was not achieved. The bot only responds to messages in #general |
| Tagging the bot or using a command in a group conversation | Partially achieved. The bot can be tagged but it won’t respond to being tagged. It is currently implemented in #general which is a group channel (even if currently only one person is in the channel) |
| Bot initiated interaction | Not achieved. |
| Reminders and reports sent to private or group chats | Partially achieved. Reports can be sent to a group chat but reminders were not implemented and neither was private messaging the bot. |
| Allow remote client connections via the internet | Achieved. |
| access SOB database | The program can access all tables in the database and return data to Slack |
| send appropriate error messages to the user/client | Partially achieved. Some error messages were implemented but not enough. |
| create and deliver reports to individual teachers that contain:   * list of their students who are behind with a list of which SOBs those students have overdue * list of students with bad attendance with dates they have not attended * table showing all their students with number of SOBs | Private messaging and teacher specific reports were not implemented. |
| Bot must be implemented in Python 2.7.14 | ✔ |
| Must use the Slack API | ✔ |
| Accessible from a multitude of devices via Slack including: Desktop Slack app, Slack in browser and Slack mobile phone/tablet app | Was proven to work in the Slack desktop app, in browser and on mobile. See testing for details. |
| Simple to use | The setup for the app requires detailed instructions and expertise, but after the setup has been completed it’s a simple matter of typing the correct commands. |
| Fast to use | Typically the system takes <5 seconds to complete a task. |
| The data must be returned after a request within 1 minute | It always is |
| Use friendly/pleasant language | All of the error messages or commands are phrased in a neutral to pleasant tone. |
| Easily maintainable | After initial setup the system needs little to no maintenance. |
| Accessible | As previously detailed the use of a text based system makes the system very accessible. |
| Student data secure | The data is very secure as to access the data 3 different verification codes are needed. Also the tools used, like Flask, are lightweight and don’t require updates, which means they don’t have many vulnerabilities to exploit and patch. |
| The user guide must be easy to understand and clearly presented | The user guide was not completed, but all information to install the whole system is in the implementation section. |

## Management of project and problems encountered

The project was supposed to start at week 5, but because of problems with the original supervisor becoming ill and having to switch to a different supervisor, the start of the project was delayed. The time spent waiting to hear from the supervisor before being told she was ill could have been spent working alone but the project subject was changed when the supervisor changed. This set the planning stage of the project back by 3 weeks. Weeks 6-12 were supposed to be used to do requirements analysis and design, but the project concept itself was still being explored up until it came time to write the interim report at week 12. The project was not planned by the supervisor or student in advance like many are, it was made up on the fly so it was less well thought out. This lead to the requirements analysis and design stages being done very quickly to meet the proposal deadline, and the implementation stage being started without proper preparation. As the implementation phase was badly planned it went slowly, and other deadlines interfered with the milestones. Unfortunately because of the delays only a proof-of-concept version of the project was implemented; many of the planned for features are absent.

The first step in creating the program was getting the SOB monitor website running locally on the project machine. This took a long time as I don’t have much experience with web development, so I didn’t know why the site wouldn’t work. Franco helped but I only asked for help after weeks of failing to fix it myself

I am used to fully fledged IDEs so working with the Flask debugger was difficult. The Flask debugger required command line interface and setting up yet another server to monitor problems. I only figured out how to use the debugger after completing the program.

### Original Project Milestones

|  |  |  |
| --- | --- | --- |
| Activity | Estimated Duration | Estimated Completion Week |
| Requirements Analysis | 4 weeks | Week 9 |
| Design | 4 weeks | Week 12 |
| Implementation | 8 weeks | Week 19 |
| Testing | 2 weeks | Week 21 |
| Evaluation | 2 weeks | Week 23 |
| Final Report | 4 weeks | Week 24 |

# Conclusion

## Reflection

The outcome of the project works well as a prototype but is not a fully functional system. It cannot be implemented in a real-world setting, which was they highest goal set for this project.

### Lessons learned

I became more proficient with Python, pip, PHP, SQL, github, using servers and ports, and using APIs.

I learned planning of a substantially sized project, even when undertaken by one person, is very important. Time management of sizeable projects needs to be observed carefully as certain parts of the project are likely to over run; leaving time between tasks and being able to adjust the milestones has been a hard but very worthwhile lesson to learn.

### If the project were to be done over

If the program was done over now, it would be coded with node.js instead of python. Slack provides a messaging API that would have been very helpful for this project but is not compatible with python.

The free version of ngrok was used, which only offers temporary URLs. Permanent URLs are available if the system was to be implemented permanently but for testing temporary URLs worked fine. The only downside was that every time the ngrok server timed out (every day or so) the new URL had to be entered into the Slack webhook page. Ideally a different tool could have been used, or ngrok could have been paid for.

The Slack webhook way of communicating with the bot user may not have been the most efficient way, if the project were done over the full range of options provided by Slack would be more heavily researched.

If I could do the milestones again I would plan around deadlines for other subjects, not just the deadlines for this module.

If I could do the project over again I would ask for help more frequently.

### Future work

Firstly the full set of requirements for the project should be met.

After this project has been fully realized it would be great to generalise the bot and distribute it to other Slack workspaces.

In the current workspace the bot could also be used to collate data about which SOBs are the most difficult and feed that data back to the staff so the know which SOBs need to be tweaked.

# Appendices

## References

Abramovich, S., Schunn, C. and Higashi, R. (2013). Are badges useful in education?: it depends upon the type of badge and expertise of learner. *Educational Technology Research and Development*, 61(2), pp.217-232.

api.slack.com. (2018). *Bot Users*. [online] Available at: https://api.slack.com/bot-users [Accessed 8 Feb. 2018].

Deci, E., Koestner, R. and Ryan, R. (1999). A meta-analytic review of experiments examining the effects of extrinsic rewards on intrinsic motivation. *Psychological Bulletin*, 125(6), pp.627-668.

Easley, D. and Ghosh, A. (2016). Incentives, Gamification, and Game Theory: An Economic Approach to Badge Design. *ACM Transactions on Economics and Computation*, 4(3), pp.1-26.

En.wikipedia.org. (2018). *Slack (software)*. [online] Available at: https://en.wikipedia.org/wiki/Slack\_(software)#cite\_note-SlackOpen-1 [Accessed 8 Feb. 2018].

Flask.pocoo.org. (2018). *Welcome | Flask (A Python Microframework)*. [online] Available at: http://flask.pocoo.org/ [Accessed 25 Apr. 2018].

Gayer, O. (2018). Understanding Bots and How They Hurt Your Business. [Blog] *incapsula.com*. Available at: https://www.incapsula.com/blog/understanding-bots-and-your-business.html [Accessed 8 Feb. 2018].

Halavais, A. (2012). A GENEALOGY OF BADGES. *Information, Communication & Society*, 15(3), pp.354-373.

Latham, G. and Seijts, G. (1999). The effects of proximal and distal goals on performance on a moderately complex task. *Journal of Organizational Behavior*, 20(4), pp.421-429.

Morgan, M. (1985). Self-monitoring of attained subgoals in private study. *Journal of Educational Psychology*, 77(6), pp.623-630.

Ngrok.com. (2018). *ngrok - secure introspectable tunnels to localhost*. [online] Available at: https://ngrok.com/product [Accessed 25 Apr. 2018].

Python.org. (2018). *Welcome to Python.org*. [online] Available at: https://www.python.org/about/ [Accessed 25 Apr. 2018].

Raimondi, F. (2016). *Student Observable Behaviour Monitor*. https://bitbucket.org/mdxmase/sobmonitor/src: Middlesex University.

Shawar, B. and Atwell, E. (2005). Using corpora in machine-learning chatbot systems. *International Journal of Corpus Linguistics*, 10(4), pp.489-516.

SourceForge. (2018). *XAMPP*. [online] Available at: https://sourceforge.net/projects/xampp/ [Accessed 25 Apr. 2018].

Stock, J. and Cervone, D. (1990). Proximal goal-setting and self-regulatory processes. *Cognitive Therapy and Research*, 14(5), pp.483-498.

Virtualenv.pypa.io. (2018). *Virtualenv — virtualenv 15.1.0 documentation*. [online] Available at: https://virtualenv.pypa.io/en/stable/ [Accessed 25 Apr. 2018].

Wakefield, J. (2018). *Microsoft chatbot goes rogue on Twitter*. [online] BBC News. Available at: http://www.bbc.co.uk/news/technology-35890188 [Accessed 8 Feb. 2018].

Zeifman, I. (2017). Bot Traffic Report 2016. [Blog] *incapsula.com*. Available at: https://www.incapsula.com/blog/bot-traffic-report-2016.html [Accessed 8 Feb. 2018].

W3.org. (2018). *Web Content Accessibility Guidelines (WCAG) 2.0*. [online] Available at: https://www.w3.org/TR/WCAG20/ [Accessed 25 Apr. 2018].

## Guide to the materials on the USB

The USB contains the python files mentioned in the report, Adam Jarzebak’s files at time of download from the git and a zip of the SOB monitor code. To run the main python file pybot1.py you will need to follow the instructions detailed in the implementation section, including setting up multiple servers and a database.

## My Code

### database\_test1.py

#### Returns:

### Pybot1.py





## Adam Jarzebak’s code at time of download

### get\_user\_list.py



### sendPrivateMsg.py



### slack\_message.py



### Requirements.txt



### README.md







### .gitignore

