PROJECT BOTFLY

Digital tools for students

PROJECT DESCRIPTION

BotFly is a project to create a Discord bot using DiscordJS. Our bot will provide students and groups with various tools to assist in communication and collaboration. Features will include participation tracking, reminders and alerts and time functions.

AlphabetEsq

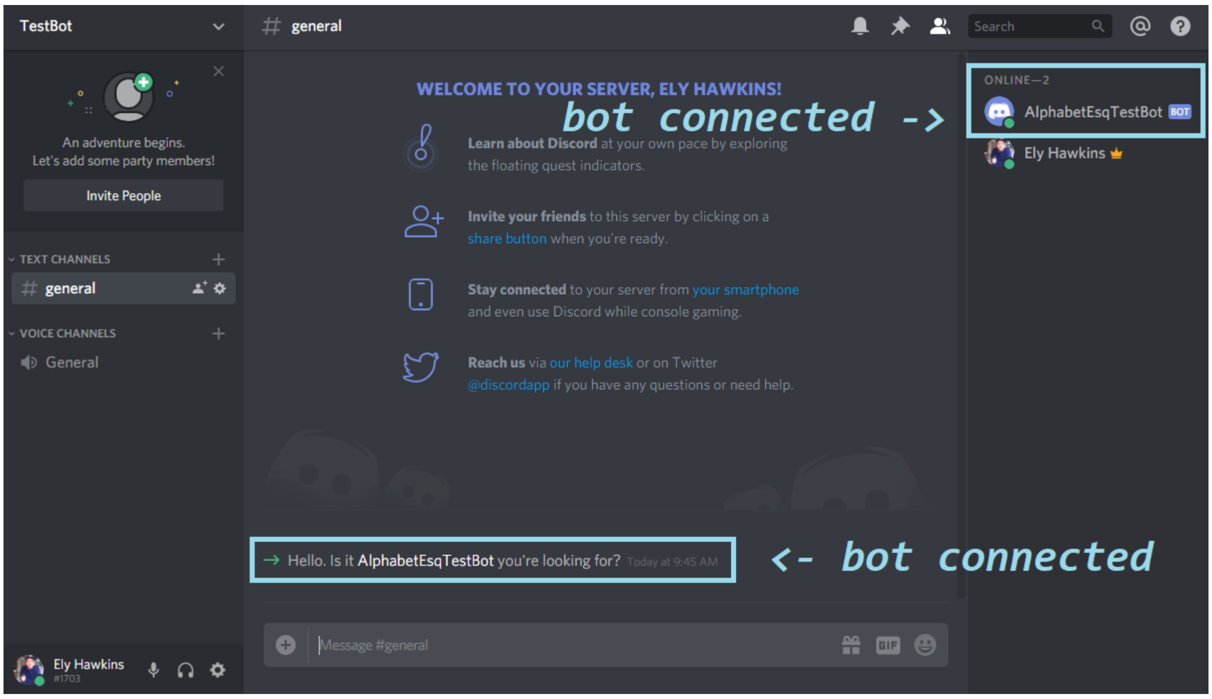
Casey Edwards | s3776107
Christopher Crawford | s3753150
Dylan Currie | s3793998
Ely Hawkins | s3754457
Melanie Broersen | s3611357

Table of Contents

1 Core Features

1.1 Core Feature 1

1.1.1 Core Feature 1 Design



WELCOME TO YOUR SERVER, ELY HAWKINS!

Learn about Discord at your own pace by exploring the floating quest indicators.

Invite your friends to this server by clicking on a share button when you're ready.

Stay connected to your server from your smartphone and even use Discord while console gaming.

Reach us via our help desk or on Twitter @discordapp if you have any questions or need help.

→ Hello. Is it AlphabetEsqTestBot you're looking for? Today at 9:45 AM

***Figure 1.1** - Bot Connection Design*

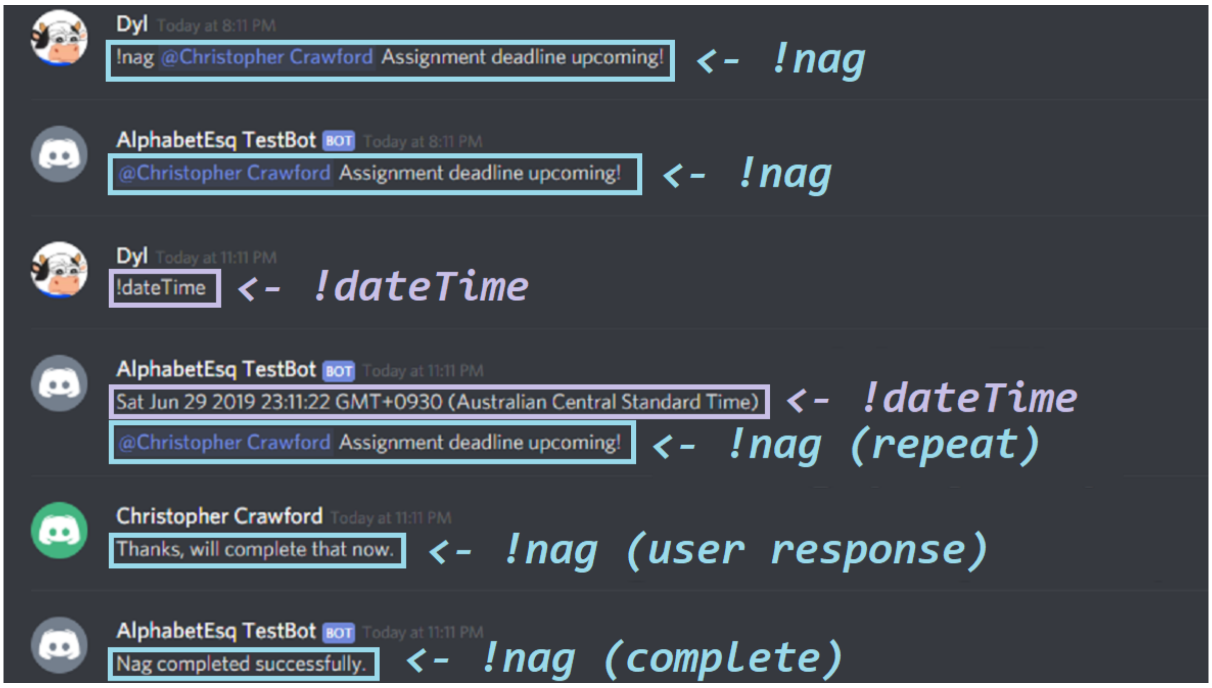
As shown in the screenshot above our bot will connect to any existing discord server. This will result in a new user connection message to the chat chanel along with the time it was imported. The bots appearance in the 'online' users to the right of the screenshot shows that it has been imported and is functioning.

1.1.2 Core Feature 1 Validation Testing:

The bot will be successfully imported into three separate test servers. Import is considered successful when connection results in a new user text notification and the Discord bot appearing among 'online users' in the right-hand panel. Validation test is successful when all authorised users can import our bot into any servers using the following steps:

1.2 Core Feature 2

1.2.1 Core Feature 2 Design



Dyl Today at 8:11 PM
!nag @Christopher Crawford Assignment deadline upcoming!

AlphabetEsq TestBot BOT Today at 8:11 PM
@Christopher Crawford Assignment deadline upcoming!

Dyl Today at 11:11 PM
!dateTime

AlphabetEsq TestBot BOT Today at 11:11 PM
Sat Jun 29 2019 23:11:22 GMT+0930 (Australian Central Standard Time)
@Christopher Crawford Assignment deadline upcoming!

Christopher Crawford Today at 11:11 PM
Thanks, will complete that now.

AlphabetEsq TestBot BOT Today at 11:11 PM
Nag completed successfully.

Figure 2.1 - *!dateTime & !nag design*

The command **!nag *@username message*** results in the accompanying message being sent every three hours until member mentioned in posts. Ceases once response is received. The nagged user is tagged in messages.

The **!dateTime** command sends the current time in the bots local timezone.

1.2.2 Core Feature 2 Validation Testing

!dateTime - Tested by three users in separate time zones. Response to be as expected in format demonstrated above, including correct information of time zone. User will type the command !dateTime and document response, noting accuracy against below.

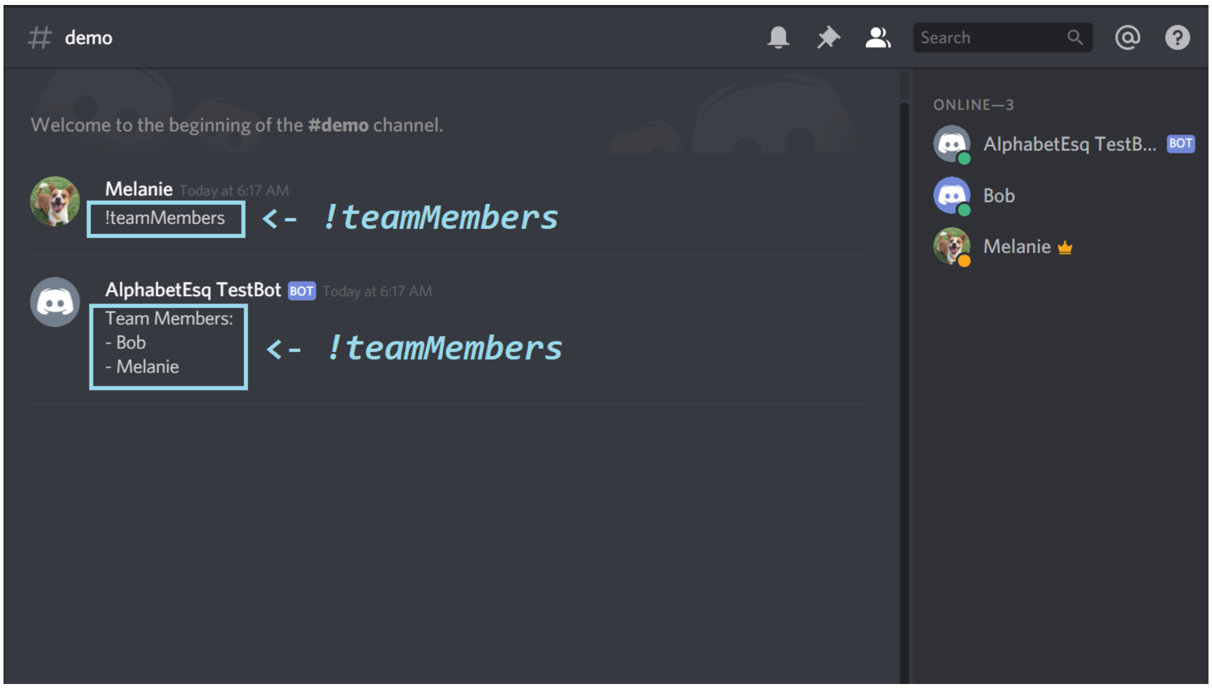
- ❑ **User - 5:00PM:** !dateTime
- ❑ BOT **- 5:00PM:** Day Mmm dd yyyy hh:mm:ss GMT+xxxx (Timezone)

!nag *@username message* - Tested by using command. Bot should mimic message exactly as provided every three hours until nagged user posts. Once user posts, bot will respond as per below and the nagging will cease.

- ❑ **Username32 - *5:00PM***: !nag @username1 Assignment deadline upcoming!
- ❑ BOT **- *5:00PM***: !nag @username1 Assignment deadline upcoming!
- ❑ BOT **- *8:00PM***: !nag @username1 Assignment deadline upcoming!
- ❑ **Username1 - *8:05PM***: Getting onto this now, thanks!
- ❑ BOT **- *8:05PM***: Nag completed successfully.

1.3 Core Feature 3

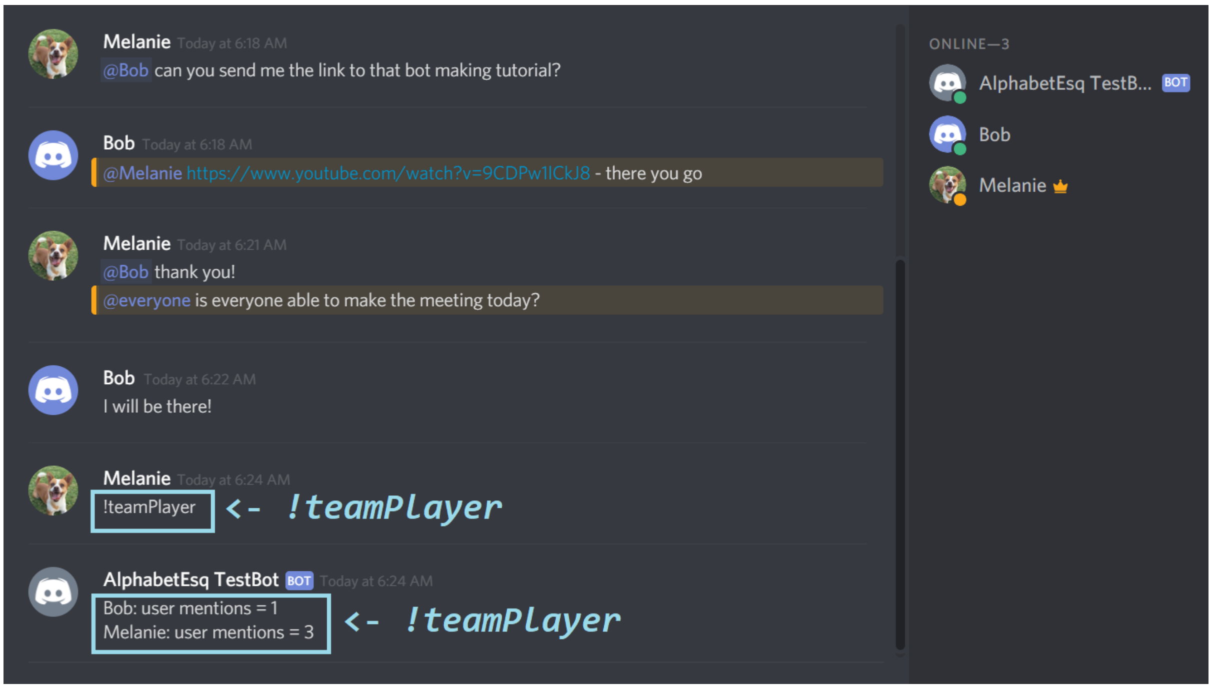
1.3.1 Core Feature 3 Design



```
# demo

Welcome to the beginning of the #demo channel.

Melanie Today at 6:17 AM
!teamMembers                <- !teamMembers

AlphabetEsq TestBot BOT Today at 6:17 AM
Team Members:
- Bob                       <- !teamMembers
- Melanie

ONLINE—3
AlphabetEsq TestB... BOT
Bob
Melanie
```

Figure 3.1 - *!teamMembers design*

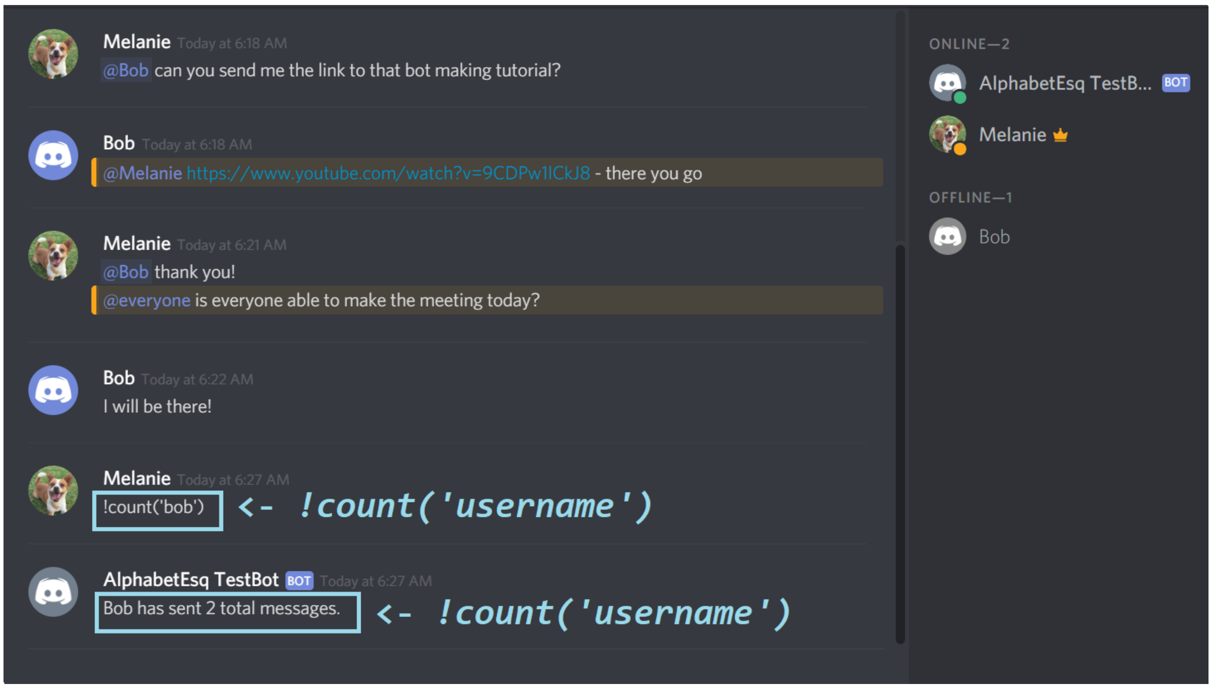
The command **!teamMembers** returns all team members (active/non-active) who have been a part of the existing chat history.



```
Melanie Today at 6:18 AM
@Bob can you send me the link to that bot making tutorial?

Bob Today at 6:18 AM
@Melanie https://www.youtube.com/watch?v=9CDPw1lCkJ8 - there you go

Melanie Today at 6:21 AM
@Bob thank you!
@everyone is everyone able to make the meeting today?

Bob Today at 6:22 AM
I will be there!

Melanie Today at 6:24 AM
!teamPlayer                 <- !teamPlayer

AlphabetEsq TestBot BOT Today at 6:24 AM
Bob: user mentions = 1      <- !teamPlayer
Melanie: user mentions = 3

ONLINE—3
AlphabetEsq TestB... BOT
Bob
Melanie
```

Figure 3.2 - *!teamPlayer design*

The command **!teamPlayer** returns all users on board, along with how many times they have mentioned other users in their comments.


Melanie Today at 6:18 AM
@Bob can you send me the link to that bot making tutorial?

Bob Today at 6:18 AM
@Melanie https://www.youtube.com/watch?v=9CDPw1lCkJ8 - there you go

Melanie Today at 6:21 AM
@Bob thank you!
@everyone is everyone able to make the meeting today?

Bob Today at 6:22 AM
I will be there!

Melanie Today at 6:27 AM
!count('bob') <- !count('username')

AlphabetEsq TestBot BOT Today at 6:27 AM
Bob has sent 2 total messages. <- !count('username')

***Figure 3.3 -** !count design*

The command **!count("username")** returns the total number of messages a person has typed in that particular chat channel.

1.3.2 Core Feature 3 Validation Testing

!teamMembers - Tested by three different users on a chat channel where there are at least 5 users existing on the server. Testing will be done to confirm users who have left the server but who have previously commented in the channel are included in the list of members. Response to be as expected in format demonstrated above. User will type the command !teamMembers and compare the listed response to a known list of users having contributed to the chat.

- ❏ **User1 - 6:17AM:** !teamMembers
- ❏ **BOT - 6:17AM:** Team Members:
    - ❏ User 1
    - ❏ User 2
    - ❏ User 3
    - ❏ Etc.

!teamPlayer - Tested by three different users on a chat channel where there are at least 5 users existing on the server. Testing will be done to confirm users who have used user mentions have correct user mention counts, against known amount of mentions. This will include @everyone mentions, which will be counted for everyone on the server - including the poster. Response to be as expected in format demonstrated above. User will type the command *!teamPlayer* and compare the list response to a known list of users and a known user mention for each.

- ❏ **User1 - 6:20AM:** *@everyone* reminder assignment deadline is coming!
- ❏ **User2 - 6:21AM:** *@User1* thanks!
- ❏ **User2 - 6:24AM:** *@User1* I've completed my tasks now!

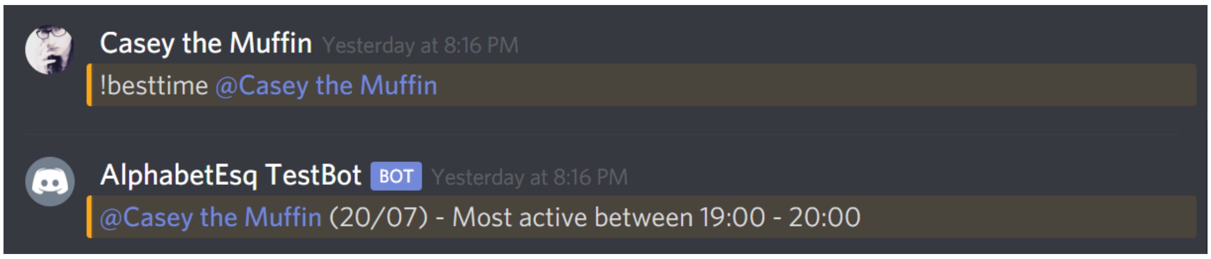
- ❑ **User1 - 6:25AM:** @*User2* Nice job!
- ❑ **User1 - 6:24AM:** !teamPlayer
- ❑ **BOT - 6:24AM:**
    - ❑ *User1:* user mentions = 3
    - ❑ *User2:* user mentions = 2
    - ❑ Etc....

!count("*username*") - Tested by three different users on a chat channel where there are at least 5 users having contributed to that channel. Testing will be done to confirm all users who have sent messages on the chat channel have correct message counts, against known amount of messages. Response to be as expected in format demonstrated above. User will type the command *!count("username")* for each user and compare the response to a known list of users and a known message count for each.

- ❑ **User1 - 6:25AM:** *Hi everyone!*
- ❑ **User2 - 6:30AM:** *Hello!*
- ❑ ***User1 - 6:32AM:*** **48 further messages**
- ❑ ***User2 - 6:33AM:*** **44 further messages**
- ❑ **User1 - 6:27AM:** !count("*user2"*)
- ❑ **BOT - 6:27AM:** *User2* has sent 45 total messages.
- ❑ **User1 - 6:30AM:** !count("*user1*")
- ❑ **BOT - 6:30AM:** *User1* has sent 50 total messages.

1.4 Core Feature 4

1.4.1 Core Feature 4 Design

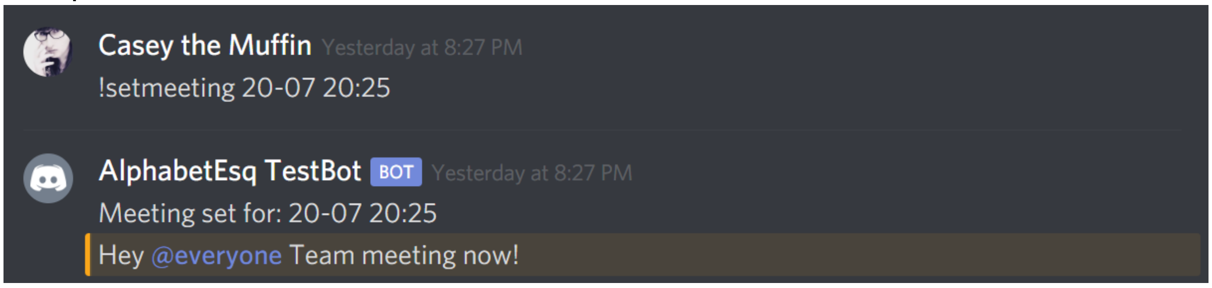


Casey the Muffin Yesterday at 8:16 PM
!besttime @Casey the Muffin

AlphabetEsq TestBot BOT Yesterday at 8:16 PM
@Casey the Muffin (20/07) - Most active between 19:00 - 20:00

Figure 3.4-1 - *!bestTime design*

The command **!bestTime @*Username*** will show the particular hour of day mentioned user posted the most.

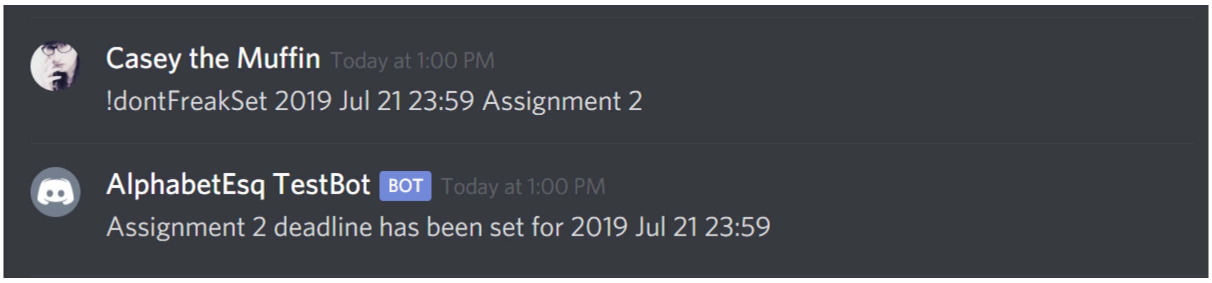


Casey the Muffin Yesterday at 8:27 PM
!setmeeting 20-07 20:25

AlphabetEsq TestBot BOT Yesterday at 8:27 PM
Meeting set for: 20-07 20:25
Hey @everyone Team meeting now!

Figure 3.4-2 - *!setMeeting design*

The command **!setMeeting *DD-MM hh:mm MeetingTitle EndTime*** stores meeting time information, sending reminders in intervals of 3 hours with a final reminder 5 minutes before the scheduled meeting time.

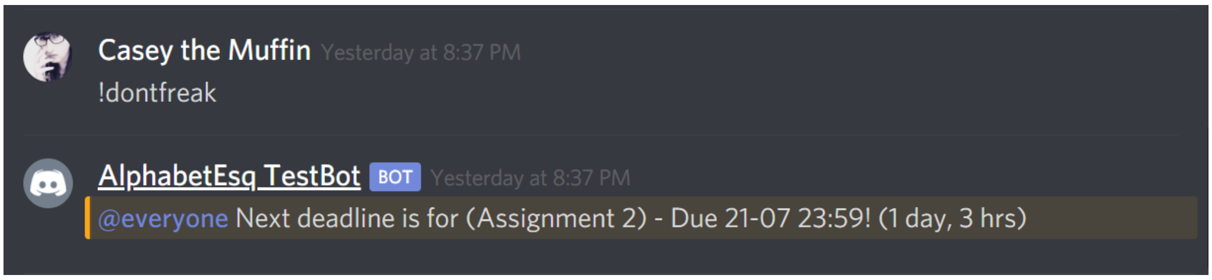


Casey the Muffin Today at 1:00 PM
!dontFreakSet 2019 Jul 21 23:59 Assignment 2

AlphabetEsq TestBot BOT Today at 1:00 PM
Assignment 2 deadline has been set for 2019 Jul 21 23:59

Figure 3.4-3 - !dontFreakSet design

The Command **!dontFreakSet *yyyy MMM dd hh:mm AssignmentTitle*** can be used to override the current dontFreak deadline with time and input provided.



Casey the Muffin Yesterday at 8:37 PM
!dontfreak

AlphabetEsq TestBot BOT Yesterday at 8:37 PM
@everyone Next deadline is for (Assignment 2) - Due 21-07 23:59! (1 day, 3 hrs)

Figure 3.4-4 - *!dontFreak design*

The command **!dontFreak** command sends a message outlining the amount of time left before the next assignment deadline.

1.4.2 Core Feature 4 Validation Testing

!bestTime - Three different users to test on separate Discord servers. Expected results would be @username, date, and the time the user posts the most. Testing results to match below example.

- ❏ **User32 7:03PM** - Hi everyone!
- ❏ **User32 8:05PM** - Working on assignment now.
- ❏ **User32 8:12PM** - Have made progress on assignment.
- ❏ **Username1 8:15PM: !bestTime @Username32**
- ❏ BOT **8:15PM -** @user32 (dd/mm/yyyy) comments most actively between 8-9PM.

!setMeeting – Feature tested using the !setMeeting command with an appropriate time beyond 3 hours in the future. Feature is validated when BotFly responds with the meeting time every 3 hours, along with a final message 5 minutes before. Test results to match below expected results.

- ❏ **User1 7:03PM** - !setMeeting 20-07 22:15 Team Meeting 22:30
- ❏ BOT **7:03PM** - Don't forget! Team Meeting: 20-07 22:15 - 22:30
- ❏ BOT **10:03PM** - Don't forget! Team Meeting: 20-07 22:15 - 22:30
- ❏ BOT **10:15PM** - 5 MINUTE WARNING! Team Meeting: 30-07 22:15 - 22:30

!dontFreakSet - Validated by successfully using the command ***!dontFreakSet yyyy MMM dd hh:mm AssignmentTitle,*** confirming the output against the below. For !dontFreakSet to be considered successfully validated, integration with !dontFreak must also be tested and validated successfully.

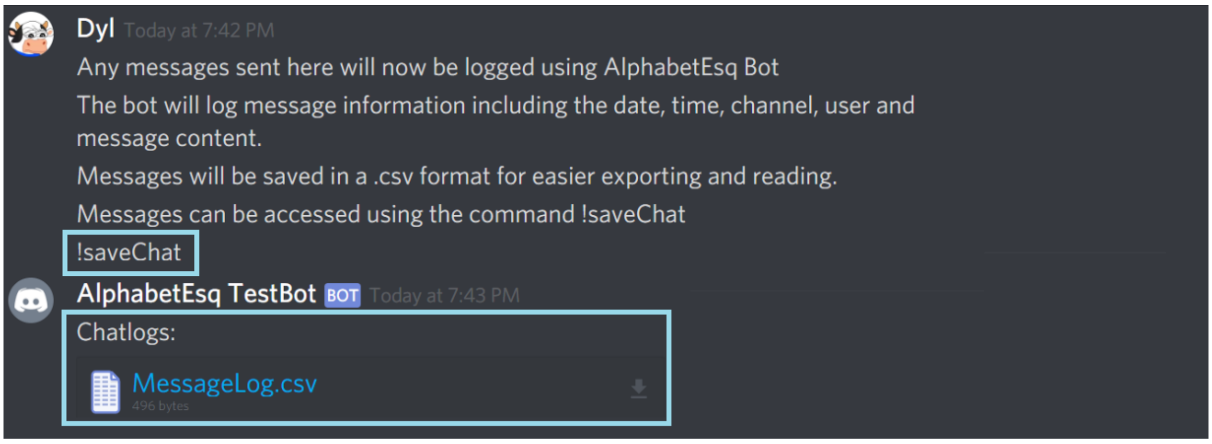
- ❏ **Username1 8:15PM -** !dontFreakSet 2019 Jul 21 23:59 Assignment 2
- ❏ BOT **8:15PM** - Assignment 2 deadline has been set for 2019 Jul 21 23:59

!dontFreak - Validated by testing command and comparing the results to !dontFreak message as set above. Output to directly print information as set in !dontFreakSet, including assignment name, due date and time left.

- ❏ **Username1 8:59PM -** !dontFreak
- ❏ BOT **8:59PM** - Next deadline is for (Assignment 2) - Due 21-07 23:59! (3 hrs)

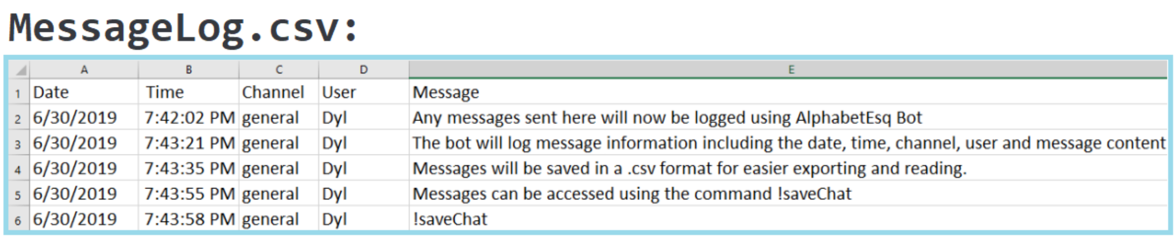
1.5 Core Feature 5

1.5.1 Core Feature 5 Design



Dyl Today at 7:42 PM
Any messages sent here will now be logged using AlphabetEsq Bot
The bot will log message information including the date, time, channel, user and message content.
Messages will be saved in a .csv format for easier exporting and reading.
Messages can be accessed using the command !saveChat
!saveChat

AlphabetEsq TestBot BOT Today at 7:43 PM
Chatlogs:
MessageLog.csv
496 bytes



MessageLog.csv:

	A	B	C	D	E
1	Date	Time	Channel	User	Message
2	6/30/2019	7:42:02 PM	general	Dyl	Any messages sent here will now be logged using AlphabetEsq Bot
3	6/30/2019	7:43:21 PM	general	Dyl	The bot will log message information including the date, time, channel, user and message content
4	6/30/2019	7:43:35 PM	general	Dyl	Messages will be saved in a .csv format for easier exporting and reading.
5	6/30/2019	7:43:55 PM	general	Dyl	Messages can be accessed using the command !saveChat
6	6/30/2019	7:43:58 PM	general	Dyl	!saveChat

Figure 5.1 - *!saveChat*

The command **!saveChat** will return a csv file called MessageLog.csv. This file will contain all messages sent while the bot is running, including information on date, time, channel, user and a copy of the message.

1.5.2 Core Feature 5 Validation Testing

!saveChat - Successful download of chat history. Chat history saved, opened and readable. Methodology documented and test is repeatable. MessageLog.csv logs messages as per expectation and demonstration.

❑ **User 8PM -** !saveChat
❑ BOT **8PM -** Chat logs: {File attachment: ***MesageLog.csv***}

MessageLog.csv **-** A csv spreadsheet containing:

❑ **Date** - Date formatted as dd/mm/yyyy
❑ **Time** - Time formatted as hh:mm:ss AM/PM
❑ **Channel** - Discord channel name
❑ **User** - Users display name
❑ **Message** - Complete message sent by user
❑ **Complete & accurate** - Accurate recording of all messages, formatted correctly.

Extended Verification - Allow the bot to run for an extended period of time before running the command. This will ensure that message logging functions over a longer period of time with a greater number of messages.

2 Project Estimation

	AlphabetEsq						
	BotFly						
{Task Name}	**Sarah Longhurst**	**Dylan Currie**	**Ely Hawkins**	**Casey Edwards**	**Christopher Crawford**	**Melanie Broersen**	**{Average Hour}**
MVF 1 - Bot Connection	3	5	4	5	5	5	5
MVF 2 - !nag & !dateTime	10	15	11	7	14	15	12
MVF 3 - Message Counter(s)	15	20	20	11	17	18	17
MVF 4 - Extended Team Functions	20	15	14	19	20	19	18
MVF 5 - !saveChatlogs	8	8	11	5	10	9	9
EF 1 - Participation Points System	18	25	26	15	15	20	20
EF 2 - Points Leaderboard	5	4	8	5	7	6	6
EF 3 - Music Player	15	12	15	10	18	16	14
EF 4 - Content Censor	12	10	9	13	8	15	11
							111

Figure 2.1 – *Project Estimation Spreadsheet*

Above is a summary spreadsheet of the project timeframe estimations by each member. Estimations are done for each function, with consideration to research, learning, designing, coding and testing time requirements.

A copy of our full project estimation spreadsheet is available below. The document contains a more detailed breakdown of our individual estimations, including information on considerations factored into calculations. Further detailed written justification is included in this report.

Full Project Estimation Spreadsheet

GoogleSheets - AlphabetEsq Estimation

2.1 MVF 1 - Bot Connection Justification

2.1.1 Casey Edwards

- Video introductions and installing software
- Getting used to the coding environment and setting up the bot
- Discord have most of the process automated it’s just a matter of setup

2.1.2 Christopher Crawford

- Lots introductory videos and tutorials on Discord Bot setup.
- DiscordJS and online guides will not make coding too difficult.
- Research and testing will not take long as everything is so well documented.

2.1.3 Dylan Currie

- DiscordJS highly documented with a vast amount of online guides available
- Research and learning would involve accessing guides/videos/etc.
- Testing is limited - if the bot connects, it works. If not, it doesn’t.

2.1.4 Ely Hawkins

- Numerous guides and videos available showing exactly how to connect a bot.
- Limited coding and testing required.
- Research time is small compared to other features.

2.1.5 Melanie Broersen

- Ample amount of tutorial videos available online on how to create and set up/connect bots.
- Very limited coding and testing required.
- Easy access to all tools required.

2.2 MVF 2 - !nag & !dateTime Justification

2.2.1 Casey Edwards

- Getting time and date to work correctly in Visual studio code.
- Research for Discord.js to program both tools in JavaScript.
- In order to test accurately long intervals will need to be tested.

2.2.2 Christopher Crawford

- Studying and researching JavaScript on how to code !nag functionality.
- Further research into Discord.js for !nag
- Testing will be a long process due to the intervals.

2.2.3 Dylan Currie

- Research and learning into JavaScript intervals required. (for !nag)
- Filtering for nag will require researching of DiscordJS documentation and/or third-party filtering methods to ensure command ends when user posts.

- Testing will need to be spread across multiple hours (6+), as such a total of 2 calculated to reflect the multiple shorter sessions.

2.2.4 Ely Hawkins

- More research into discord.js functionality necessary to complete these prompts (especially !nag).
- Long validation process.
- Use of different classes such as commando and dateTime.

2.2.5 Melanie Broersen

- !dateTime requires limited coding and research into Date object.
- nag! requires extended coding and research into setInterval method.
- Validation process for nag! requires waiting 3 hours for each nag! message, which will accumulate to 6+ hours for some tests.

2.3 MVF 3 - Message Counter(s) Justification

2.3.1 Casey Edwards

- Research on how to count out the messages accurately
- Java knowledge can be applied but will need some research for implementation into JavaScript
- Needs to be tested over a period of time to ensure no overflow occurs in the code

2.3.2 Christopher Crawford

- Struggled with Java will need to re-learn basic functions again.
- Understanding how to retrieve the data from Discord Server.
- Extrapolating data may prove time consuming.

2.3.3 Dylan Currie

- Message counter functions may require a local message cache - this could take some time to research, learn and code.
- Existing Java knowledge will prove beneficial for coding of basic loops/etc
- DiscordJS has functionality for pulling information of users on the server.

2.3.4 Ely Hawkins

- Existing Java coding knowledge might be more useful here than other features.
- May be able to share code with MVF 5
- Some counter functions may require further research.

2.3.5 Melanie Broersen

- Research into user data retrieval needed - Discord.js provides this information.
- Message caching required – can share this code with MVF5.
- Previous experience with Java will prove beneficial.

2.4 MVF 4 - Extended Team Functions Justification

2.4.1 Casey Edwards

- Particular parts of these features do benefit from code reusability.
- !BestTime will be the most difficult to code out as will need to compare data and this would normally be done using loops which are not readily available using Discord.js.
- Feature will require the most time due to it being the most functions.

2.4.2 Christopher Crawford

- Code reusability will prove helpful.
- MVF 1 and MVF 3 will help with planning, coding and the data extraction for the MVF 4 functions.
- 3 different functions will take time to complete.

2.4.3 Dylan Currie

- The !setMeeting function could reuse interval code from MVF 2 - !nag.
- The !bestTime function could pull data from the cache array created in MVF 3.
- The !dontFreak command would be similar code to the !nag command.

2.4.4 Ely Hawkins

- Code reusability between prompts.
- My personal shortcomings in programming may hinder a longer feature like this.
- Limited documentation available makes research and learning more laborious.

2.4.5 Melanie Broersen

- !setMeeting and !dontFreak features can reuse code from MVF 2.
- !bestTime will be able to use the data from MVF3.
- !bestTime will take longer to implement and test.

2.5 MVF 5 - !saveChatlogs Justification

2.5.1 Casey Edwards

- Pretty straight forward code implementation.
- Most of the time spent will be devoted to researching how to output a file for download.
- Only 1 function; should not take long to complete.

2.5.2 Christopher Crawford

- Learning how to extract CSV files in Discord.js
- Setting a file location for the data.
- Should be easier to complete due to the previous core functions.

2.5.3 Dylan Currie

- The ‘messages’ cache will already be created in MVF 3. Messages can be pulled from here.
- Saving to CSV files appears to be quite a simple task in JavaScript.
- The array of messages will already be created in MVF 3.

2.5.4 Ely Hawkins

- May be able to share code with MVF 3.
- Downloading/exporting chat logs to a document will require further research and learning.
- No other examples of bots demonstrating this feature to look at and learn from.

2.5.5 Melanie Broersen

- Message caching required – can share this code with MVF3.
- Research into file exports from server required.
- Feature only includes a single command/function.

2.6 Extended Feature 1 - Participation Pts System Justification

2.6.1 Casey Edwards

- Given that this data will need to be stored in one way or another this system will potentially take the longest to implement as there is no prior knowledge present on how this can be done.
- Possible to reuse code from some of our MVFs.
- Displaying results yet to be determined.

2.6.2 Christopher Crawford

- Previous MVF’s will help coding.
- Core research and learning completed from previous functions.
- The design on how the data will be displayed maybe the most complex part of this feature.

2.6.3 Dylan Currie

- Message counting tools already used in previous MVFs (easy to reuse / no extra research or learning required). Not too complex to code.
- Designing of ways to earn point may be timely.
- Designing and formatting of output could be timely.

2.6.4 Ely Hawkins

- High code reusability.
- A lot of design required around how the points will be stored and allocated.
- Fairly unique idea. Not many existing examples to learn from.

2.6.5 Melanie Broersen

- Research into collecting data from voice chat required.
- Message counting will reuse code from MVF3 and 5.
- Research into how the points will be stored – and further design on display.

2.7 Extended Feature 2 - Points Leaderboard Justification

2.7.1 Casey Edwards

- Display will need to be implemented in a fun artistic way.
- Long testing phase due to the way points are accumulated.
- Knowledge on how to fully implement not yet researched.

2.7.2 Christopher Crawford

- Creative elements on how the leaderboard will be displayed.
- Testing may prove long and arduous due to the manual nature of the testing.
- Use of design software.

2.7.3 Dylan Currie

- Points system already coded on EF1. Data would simply need to be presented in a new format (leaderboard format) in EF2.
- Some knowledge of array sorting algorithms required.
- Some design requirements to present data in a neat format.

2.7.4 Ely Hawkins

- A lot of code can be copied from EF1.
- Design will be most time consuming part. How to display leaderboard and how many users will be included. Do points reset? etc.
- Need to be creative in how its displayed to further incentivise participation.

2.7.5 Melanie Broersen

- Most of the code needed can be reused from EF1.
- Design for leaderboard layout required.
- Research into array handling required, to assist and refresh any existing knowledge.

2.8 Extended Feature 3 - Music Player Justification

2.8.1 Casey Edwards

- Not many guides on creating a player the way we would like.
- Will need to research how to implement and also possibly have files to play stored.
- Inexperience with the code style may impede progress.

2.8.2 Christopher Crawford

- Research and understanding how voice commands work in discord.js.
- Understand how music players function in Discord and how to be coded.
- My inexperience with coding may take longer to finish feature.

2.8.3 Dylan Currie

- Music players are a very common Discord bot feature with many well documented guides and tutorials.
- Some research required on methods of setting up encoders or any other required tools for the Music Bot to function.
- Testing of all features may take some time.

2.8.4 Ely Hawkins

- Research suggests to coding side of things could be difficult.
- Vast array of guides and tutorials.
- More complex coding may be difficult for me and thus increase time needed.

2.8.5 Melanie Broersen

- Plentiful amounts of guides online to assist with design.
- Research into how to integrate YouTube in Discord required.
- Coding will be more complex, will need to refresh memory and research how to implement code.

2.9 Extended Feature 4 - Content Censor Justification

2.9.1 Casey Edwards

- Very simple implementation, many examples around.
- Bot will only need to detect a word and block it.
- Can either censor the word, or delete the message the word is in. Either has fairly simple implementation.

2.9.2 Christopher Crawford

- May take time to decide decipher what content has to be blocked.
- Should not be that complex for the Bot to detect censored content.
- Feature will take time to complete because of ongoing blocked content.

2.9.3 Dylan Currie

- Research into content censor dictionaries required, included licensing.
- on.message is already used in existing functions - censor could be added there.
- Reading of DiscordJS documentation required for understanding process of editing, deleting or managing other users messages required.

2.9.4 Ely Hawkins

- Plenty of examples and documentation to learn from.

- Design discussion required around what type of content we want to block.
- Fairly simple to test and validate.

2.9.5 Melanie Broersen

- List of content planned on filtering required for design.
- Research into which content dictionaries required for our specific design.
- Research into how to manage/delete other user's messages - simple process.

3 Listing Technologies

3.1 Collaborative workspaces

3.1.1 Google Docs

Google Docs allows for live collaboration on documents, ensuring team members can work together in the creation of documentation. Throughout this project GoogleDocs will be used for the creation, version control and collaboration of documentation.

Google Docs Link: Google Drive - Building IT Systems
Google Docs Guide / User Documentation: Google Docs - Docs Editors Help

3.1.2 Trello

Trello is specifically designed along Kanban board principles in order to assist teams in efficient delivery of its objectives. Trello's power comes from instant update across geographically remote users. Our team will make use of Trello to ensure all team members are aware of their responsibilities and assigned tasks. Team members can use this space to collaboratively document progress and provide updates on the status of on-going tasks.

Trello link: AlphabetEsq Trello Board
Trello Guide: Getting Started with Trello

3.1.3 Discord

Discord will be used as our primary means of communication throughout this project. Text channel(s) will be used for announcements and discussion, with weekly meetings held in voice channels.

***Discord Link*:** AlphabetEsq Discord

3.1.4 Github

Our team will use of Github for collaborative software development, making use of revision and version control for ease of testing. Github integrates well with Visual Studio Code, allowing for automatic version control and syncing.

Github link: AlphabetEsq Github
Access Guide: AlphabetEsq Github Clone Guide
General User Docs: Github Guides

3.2 Software

3.2.1 Visual Studio Code

Visual Studio Code is the primary piece of software we will use to build our bot. It is a source-code editor that provides a workspace to build and debug (web) applications. Key features include:

- In-built Git functionality which allows us to share and sync code to GitHub.
- Allows for the coding of JavaScript outside of the browser through the use of NodeJS.
- Highly customisable with a vast array of extensions.
- Simple user friendly layout and design.

Visual Studio Code Link: Visual Studio Code
Documentation: Visual Studio Code Documentation
Version: 1.35

3.2.2 Discord

Discord is a voice over IP and messaging application, that was initially designed for the gaming community as a reliable third party source for in game communication. Not only is it the platform for which our bot is being constructed, it is also our group's primary means of communication.
Key features include:

- Multi platform (desktop and mobile). Many of our group members work so being able to read and send messages from a smartphone is very important for maintaining productivity.
- Specialised communication channels for large groups such as text, image, video and audio.
- Gives us the ability to create test bots and servers to error check and stress test our code.
- Simple user friendly layout and design.
- Provides us with a direct user interface to test our applications outputs.

Discord Link: BITS-OUTA-SP2 Discord
Version: 6

3.2.3 Git-scm

Git-scm allows us to integrate Git into Visual Studio Code. Without it we would need to use both applications separately. It gives us access to the following Git functions without leaving VS Code:

- Initialize a repository.
- Clone a repository.
- Create branches and tags.
- Stage and commit changes.
- Push/pull/sync with a remote branch.
- Resolve merge conflicts and view differences.

Git-scm Link: Git-scm
Documentiation: Git-scm Documentation
Version: 2.22.0

3.2.4 Node.js

Node.js is a JavaScript runtime allowing for JavaScript to be used outside of a browser. When used in conjunction with Visual Studio Code, Node.js allows developers to instantly run and test code with ease. Through the use of Node.js, our team will be able to develop and test our Discord bot without any issues involved in the use of browsers or other JavaScript runtimes.

- Increased compatibility (compared to browsers)
- Integrates with Visual Studio Code, allowing for seamless testing

Node.js Link: Node.js
Documentiation: Node.js Docs
Version: 10.16.0 LTS

3.2.5 Discord.js

Discord.js will serve as the core of our project. Discord.js is a node.js module that allows developers to interest with the Discord API. With extensive features and comprehensive user documentation, Discord.js will be very beneficial for the success of our project.

Discord.js Link: Discord.js
Documentiation: Discord.js Documentation
Version: v11.5

3.3 Tools

3.3.1 Discord Developer Portal (Discord API)

Discord Developer Portal provides a well documented and extensive set of tools for the development of applications which work with Discord API. It is the most accessible and user-friendly way to develop applications using the Discord API. The portal contains various features allowing for easier collaboration and development of bots. The ‘Teams’ feature will be used to ensure all members have the required access and permissions.

Discord Developer Portal Link: Discord Developer Portal
User documentation: Discord Developer User Documentation
Version: 6

3.4 Resources

3.4.1 DiscordJS / Discord Bot Documentation

DiscordJS has extensive documentation available for developers. Documentation is available on every class, function and variable. Documentation will be used for researching and learning about various features and functions unique to DiscordJS.

DiscordJS Documentation: Discord JS Docs
DiscordJS Tutorial: JavaScript Discord Bot Tutorial
Discord Bot Setup (text): How to make a Discord bot
Discord Bot Setup (video): Code a JS Discord Bot: Pt 1 - Getting Set Up
Discord Commando (video): Code a JS Discord Bot: Pt 2 - Setting up Commands

3.4.1 YouTube

YouTube is host to many video tutorials and resources which will prove helpful in the learning stages of development. Videos are available for setting up Discord bots, tutorials on Discord Commando and general JavaScript tutorials amongst other things.

Discord Bot Setup (video): Code a JS Discord Bot: Pt 1 - Getting Set Up
Discord Commando (video): Code a JS Discord Bot: Pt 2 - Setting up Commands

3.4.3 Lynda

Lynda is a resource with an abundance of information on a wide range of topics. As RMIT students, all team members have unlimited access to Lynda training. Our team may make particular use of training in areas such as JavaScript.

JavaScript Tutorial: JavaScript Essential Training

3.4.4 Github Learn

Github Learn provides various lessons and challenges aimed at allowing new users to learn to use Github and its various features. As our team will be making extensive use of Github throughout our assignment, it's important that all members have an understanding of the program.

Github Learn: Resources to Learn Git

3.4.5 Visual Studio Code Docs

Visual Studio Docs provides various guides for setting up and getting started with Visual Studio Code. As our team will make use of this program for all of our collaborative coding work, a great understanding of its features is essential. The documentation section has guides on various topics, including node.js.

Visual Studio Code Docs: Getting started with Visual Studio Code
Visual Studio Code Setup Guide: Setting up Visual Studio Code
Visual Studio Code Intro Videos: Visual Studio Code Introductory Videos
Visual Studio nodeJS Guide: Build node.js Apps with Visual Studio Code