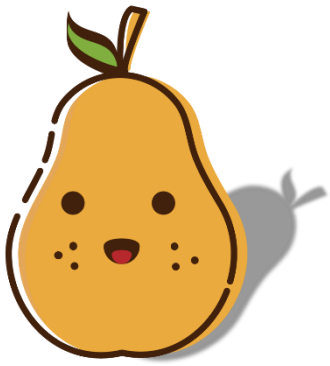
CST8333

Assignment 03



Pear

A Discord User Pairing Bot

Kaitlyn Gatineau

August 11th, 2022

Table of Contents

Introduction

As the community manager of a Discord group called the *Femmes and Thems in Tech*, I oversee, engage with and host events for thousands of spectacular women and nonbinary users who utilize our platform daily. Every week, it is my duty to manually pair individuals up who opt in to "break the ice" for those participating and allow for users to meet someone new every week.

Pear is a tool that was not only created for the sake of a school project but serves as the first iteration of a solution to a real-life problem in our community. Manually pairing users worked on a small scale, but the process needed automation and the type of tool necessary to make this happen did not exist at the time of writing for the Discord platform.

Automating the pairing process using a Discord bot not only saves time for the moderation team, but it allows for accuracy and consistency for when pairings would be posted.

While the bot will be originally utilized to pair people up for networking, it is intended to work as a pairing system for any scenario a Discord admin may deem necessary. Our goal is to create a versatile pairing bot that pairs users for any social/logistical need.

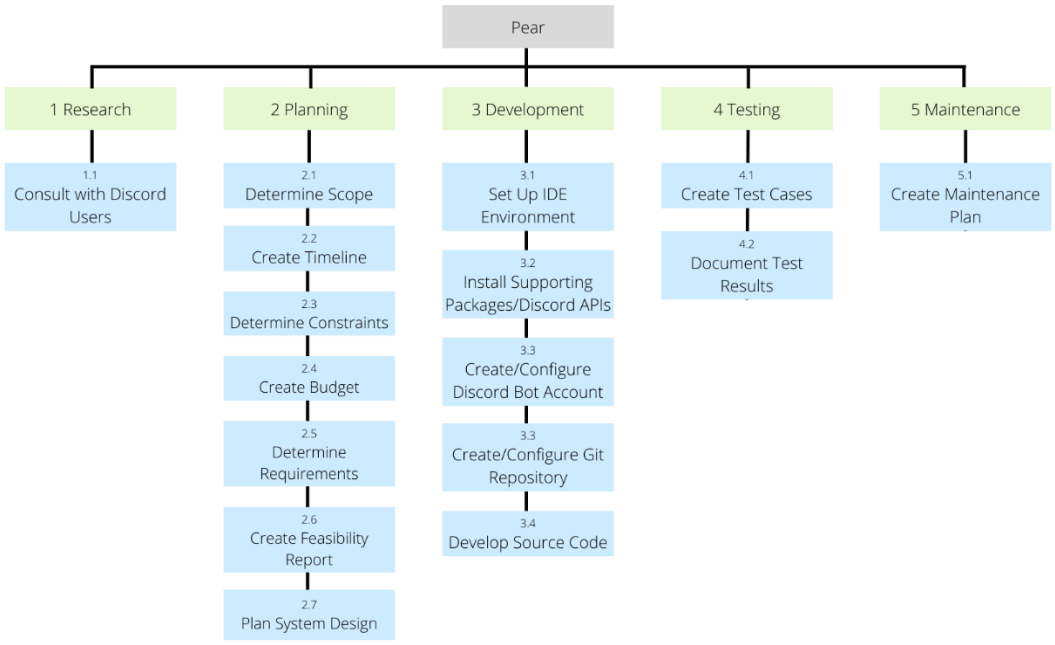
Approach

This project (in respect to the version being created for the purpose of this course) is being conducted using a modified waterfall lifecycle. The project will be done in three phases, each in respect to the assignment/presentation due at the end of each phase.

The measurement criteria for each phase will be the checklists/rubrics associated with the assignment/presentation. 60% of the source code must be completed by the end of phase two, and 100% completed by the end of phase three. The phase will be deemed successful should the majority of checklist/rubric goals for that phase be met.

Work Breakdown Structure

Below the WBS for this project. This is subject to change should new needs arise.



Version Control

Version control will be maintained using a Git repository. I plan to utilize the GitHub platform to maintain the code.

Objective

The overall objective is to create a function bot that works as intended by pairing users upon request. This main objective can be broken down into the following:

Table 1: Objectives and Business Outcomes

No.	OBJECTIVE	BUSINESS OUTCOME	MEASUREMENT CRITERIA
1	Bot only pairs users who opt into the pairing system versus all users/online users in the Discord.	Users who wish to be paired are paired, and those who do not wish to be paired are not bothered by the bot.	Users who are opted in are pinged (@ tagged); users who are not are left alone.
2	Bot is only responsive to commands made by Administrator ranks.	Users can enjoy the pairing system without unnecessary/duplicate pairings triggered by unauthorized users.	User with Administrator rank can trigger the bot via command; Bot does not respond to general user commands.

No.	OBJECTIVE	BUSINESS OUTCOME	MEASUREMENT CRITERIA
		Administrators can rely on the bot to not populate the server with unwanted posts.	
3	Bot activity is limited to the designated channel.	Users know where to look for the postings with ease.	Bot only triggers from commands and posts in a designated channel.
4	Bot fetches user IDs from one designated post only.	Users who reacted to unrelated posts in the Discord are not accidently considered in the pairings.	Bot only pairs users who react to a single post. Users who react to other posts are not included.

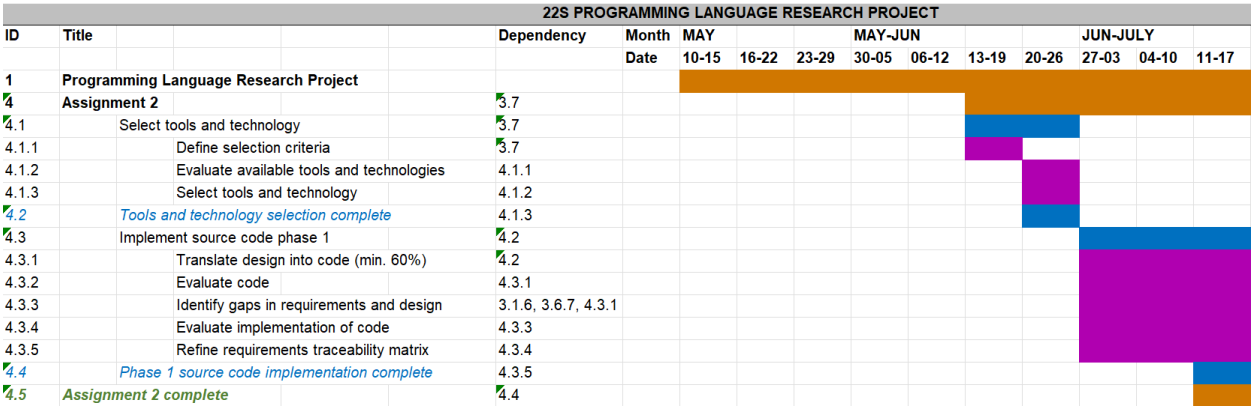
Scope

This section describes the features of the bot that will be implemented for the scope of this course. Features beyond the scope are listed as “Activities Out of Scope”.

Table 2: Project Scope

ACTIVITIES IN SCOPE	ACTIVITIES OUT OF SCOPE
1. Assignment 1	1. Apply for business financing
2. Fetch only user IDs/usernames of users who “reacted” to a designated post to be entered into the pairing pool to ensure only those who opt in are paired.	2. Store pairing data in a database schema
3. Randomize the collection order to ensure users are not paired by their place on the “reaction” list.	3. Ensure no two users are paired multiple weeks in a row
4. Ensure the post can only be triggered by an “Administrator” rank of the Discord server.	4. Create logic for the bot to post at a designated time without the need to be manually triggered by command.
5. Create a mock Discord server for development with minimum six users for testing.	5. Purchase a hosting service to maintain high availability uptime without the need for an admin to host the bot on a personal machine
6. Create a GitHub repository to maintain the code	
7. Remove reactions by users who have left the Discord.	

Timeline



22S PROGRAMMING LANGUAGE RESEARCH PROJECT																	
ID	Title			Dependency	Month	MAY			MAY-JUN					JUN-JULY			
					Date	10-15	16-22	23-29	30-05	06-12	13-19	20-26	27-03	04-10	11-17		
1	Programming Language Research Project																
4	Assignment 2			3.7													
4.1		Select tools and technology		3.7													
4.1.1			Define selection criteria	3.7													
4.1.2			Evaluate available tools and technologies	4.1.1													
4.1.3			Select tools and technology	4.1.2													
4.2		*Tools and technology selection complete*		4.1.3													
4.3		Implement source code phase 1		4.2													
4.3.1			Translate design into code (min. 60%)	4.2													
4.3.2			Evaluate code	4.3.1													
4.3.3			Identify gaps in requirements and design	3.1.6, 3.6.7, 4.3.1													
4.3.4			Evaluate implementation of code	4.3.3													
4.3.5			Refine requirements traceability matrix	4.3.4													
4.4		*Phase 1 source code implementation complete*		4.3.5													
4.5	*Assignment 2 complete*			4.4													

Clicking the icon below will open the full version in Excel. Please note, this Gantt Chart was created using the template provided, and is cited as a source in the *References*.

CST8333 Gantt Chart - Kaitlyn Gatin

Milestones and Deliverables

This section lists major milestones/deadlines required by this project.

Table 4: Project Milestones and Deliverables

MILESTONE	DATE	DELIVERABLES
1. Assignment 1 - Complete	June 9 2022	Written report and slide presentation
2. Assignment 2 - Complete	July 14 2022	Written report, slide presentation and 60% of source code
3. Assignment 3 - Complete	August 11 2022	Written report, slide presentation and 100% of source code

Risks

This section mentions the risks that are associated with this project.

Table 5: Project Risks (example)

No.	RISK DESCRIPTION	PROBABILITY (H/M/L)	IMPACT (H/M/L)	MITIGATION
1.	Schedule slippage	M	H	Track project scope and timeline

No.	RISK DESCRIPTION	PROBABILITY (H/M/L)	IMPACT (H/M/L)	MITIGATION
2.	Discord server outage	L	M	Wait until Discord resolves the issue(s)
3.	Hardware Malfunction	L	H	Back up project to external source; have a spare machine ready
4.	Lack of Knowledge	M	M	Reach out for help from the internet/experienced developers

Assumptions

Table 6: Project Assumptions

No.	THE FOLLOWING IS ASSUMNED
1.	Fundamentals of new programming language will be learned and put to use, timely, to complete project
2.	Bot will only run in a single server, via a single host machine at any given time
3.	Tracking previous pairings is not necessary for the scope of this project

Technical Constraints

Table 7: Technical Constraints

No.	TECHNICAL CONSTRAINTS
1.	Program requires a user to execute on a host computer for it to function

Budget

Below is the proposed budget for the project. The project is measured in “Level of Effort” instead of currency as no money will be spent on this project.

Please note, the Project Budget was created using the template provided, and is cited as a source in the *References*.

Table 8: Project Budget

ID	Title	Level of Effort (Days)
1	**Programming Language Research Project**	57.20
2	**Planning**	5.00
2.1	Review project approach	1.00

2.2		Define project scope	1.00
2.3		Identify project risks, assumptions and constraints	1.00
2.4		Draft project budget	1.00
2.5		Document project schedule (timeline)	1.00
2.6	***Planning Complete***		0.00
3	**Assignment 1**		12.50
3.1		Gather requirements	5.00
3.1.1		Define requirements gathering process	1.00
3.1.2		Develop use cases	1.00
3.1.3		Describe data requirements	0.50
3.1.4		Document technical requirements	0.50
3.1.5		Illustrate user interface	1.00
3.1.6		Create requirements traceability matrix	1.00
3.2		*Requirements gathering complete*	0.00
3.3		Complete feasibility study	4.00
3.3.1		Define evaluation criteria	1.00
3.3.2		Analyze development options	1.00
3.3.3		Complete cost/benefit analysis	1.00
3.3.4		Document recommended solution	1.00
3.4		*Feasibility study complete*	0.00
3.5		Provide design description	3.50
3.5.1		Create high level design	1.00
3.5.2		Develop prototype	1.00
3.5.3		Refine requirements traceability matrix	0.50
3.5.4		Draft deployment plan	0.50
3.5.5		Draft support plan	0.50
3.6		*Design description complete*	0.00
3.7	***Assignment 1 complete***		0.00
4	**Assignment 2**		19.00
4.1		Select tools and technology	7.00
4.1.1		Define selection criteria	3.00
4.1.2		Evaluate available tools and technologies	3.00
4.1.3		Select tools and technology	1.00
4.2		*Tools and technology selection complete*	0.00
4.3		Implement source code phase 1	12.00
4.3.1		Translate design into code (min. 60%)	3.00
4.3.2		Evaluate code	2.00
4.3.3		Identify gaps in requirements and design	2.00
4.3.4		Evaluate implementation of code	3.00
4.3.5		Refine requirements traceability matrix	2.00
4.4		*Phase 1 source code implementation complete*	0.00
4.5	***Assignment 2 complete***		0.00
5	**Assignment 3**		20.70
5.1		Implement source code phase 2	5.00
5.1.1		Translate design into code (up to 100%)	1.00
5.1.2		Evaluate code	1.00
5.1.3		Identify gaps in requirements and design	1.00
5.1.4		Evaluate implementation of code	1.00
5.1.5		Refine requirements traceability matrix	1.00
5.2		*Phase 2 source code implementation complete*	0.00
5.3		Complete testing	6.20
5.3.1		Refine test plan (traceability matrix)	0.40

5.3.2		Prepare test tracking tools	0.40
5.3.3		Create test environment	0.40
5.3.4		Complete system testing	0.40
5.3.5		Implement build procedures	0.40
5.3.6		Load data into test environment	0.40
5.3.7		Develop test scripts	0.40
5.3.8		Execute test scripts	1.40
5.3.9		Correct defects (if any)	0.40
5.3.10		Document test results	0.40
5.3.11		Obtain approval of testing from Facilitator	0.40
5.3.12		Update support documents (if required)	0.40
5.3.13		Store test data	0.40
5.4	*Testing complete*		0.00
5.5	Complete deployment		6.00
5.5.1		Refine deployment plan	1.00
5.5.2		Prepare production environment	1.00
5.5.3		Complete data conversion activities	1.00
5.5.4		Complete operation and maintenance prep	1.00
5.5.5		Validate code functionality	1.00
5.5.6		Deploy code	1.00
5.6	*Deployment complete*		0.00
5.7	Complete presentation		4.50
5.7.1		Select slide presentation template	1.50
5.7.2		Develop presentation content	1.50
5.7.3		Submit presentation to Facilitator	1.50
5.8	*Presentation complete*		0.00
5.9	***Assignment 3 complete***		0.00
5.10	***Programming Language Research Project complete***		0.00

Status

This is the Status Report, also known as Progress Report, for this phase of the project.

Introduction

This report gives a progress update on the accomplishments, goals, roadblocks and lessons learned over the span of Assignment 3.

Accomplishments

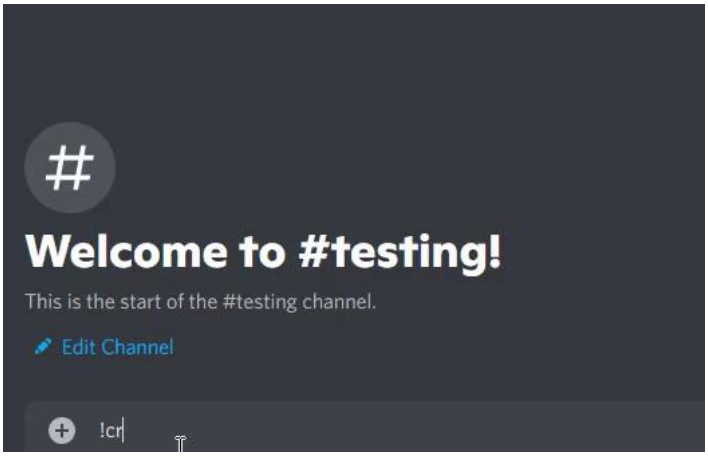
Below are my accomplishments for this assignment.

Completed Source Code

This iteration, I have completed the source code for the scope of this course. The original version (before refactoring) was finished ahead of schedule.

Increased User Friendliness

Originally, the bot was difficult to initiate because it required many parameters within quotation marks to execute the !create command. A goal I had set was to better the user experience by creating a prompt loop that accepted each input one by one instead of as parameters. This is shown below:



Goals

Below are the goals I achieved during this assignment.

Code Cleanup/Refactoring

Originally, the Discord bot was written in one large file, bot.py. This proved to be tedious to keep up with and against the better practices of Python development. Another goal I achieved was to clean up the code for better readability. While it is still far from perfect, it is easier to handle (and test) than it was in one large file.

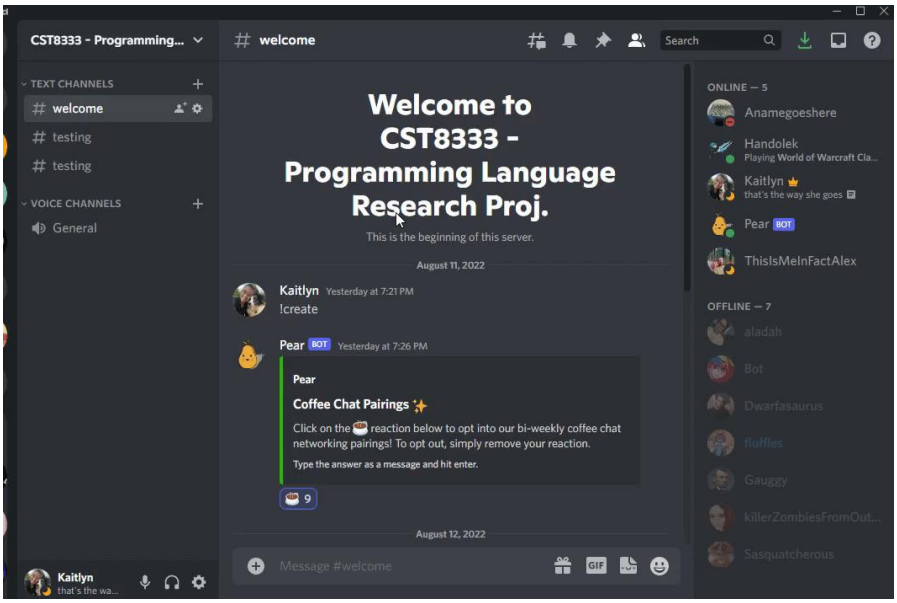
Implement Testing

As mentioned above, implementing testing was an essential goal for this phase that I have accomplished. That being said, it took a lot of time and effort to figure out how to implement unit testing within my project to add automation to my testing environment, but I managed to do so.

Deployment Preparation

The last goal worth mentioning is preparation for deployment. While more logic and database connectivity will be required to utilize this bot in my community Discord, for the scope of this project, we have began the process of deployment.

The bot will not be deployed on a server for the scope of this project, but will be available at https://github.com/kgatineau/pear_discord_bot . Clicking on the “Pear” title in any of the embeds will also redirect a user to the Github repository.



Roadblocks

Below are the roadblocks I faced during this phase of the assignment.

Unit Testing

Unit testing was the hardest roadblock to overcome and the most time consuming. I spent many days from sun-up to sun-down trying to figure out how to get the async Discord API command functions to work with unit testing, with no progress.

I came across this article stating how it was easier to pull logic out of the bot commands and into separate functions. Spent an entire day figuring out how to make unit testing work for the refactored methods before realizing that I needed to create class objects within the test.py file itself to mimic a test Discord user and a test Discord message object. However, I was unable to succeffully create a mock for the Discord context, but was able to work around it by modifying the source code by removing the “ctx” parameter in the set_pair_embed command located in bot_functions.py.

Required Code Refactorization

While refactoring the code from one large file into two files yielded optimal results, the refactorization itself took a lot of time that I did not plan for. Even after taking the time to refactor thinking this would

solve my problems creating unit test cases, I ended up still struggling mimicking the context needed to replicate Discord functions.

Lessons Learned

Below are the lessons learned throughout this phase of the project.

Create Test Cases First

In what's known as Test-Driven Development, test cases (or the outline of test cases) are written and implemented before the code is. This forces the developer to prioritize testing and create code that is compatible with the testing structure.

Writing a Discord bot is easy when writing code without acknowledging unit testing. Commands are written as their own functions and annotated as such. The logic for each command is placed in the block of code designated for the command and then you are set.

Had I done more research on unit testing and thought ahead while creating the bot, I would have written the code in a way where it could have been tested the first time around, and would have known the limitations of testing before spending an ample amount of time trying to "make it work".

Over-Commenting

Previously, I submitted code that had way too many comments. I have a habit of commenting every tiny bit of code, which over-saturated my project and learned recently that every line of code does not need to be decorated with comments.

Smaller/Multiple Files

Writing everything in one large file is not optimal for Python projects. My code reached over 200 lines of code before I realized that I should have broken the program up into separate files. After doing a bit of research on how to accomplish this, I learned that importing methods from a second Python file is easier than originally anticipated.

Don't Think Too Far Ahead

A large challenge for this phase was prioritization of events for this project. As you know, I am writing this bot for a personal community I manage and I have been chatting with other developers and community members on how to store pair data, how to create different pair sizes and other features that were out of scope for this project.

Source Code Implementation

Introduction

The language selected for this project is Python. Python is a high-level programming language that is interpreted instead of compiled and has a dynamic range of syntax usages. Python is a very versatile language and is user friendly relative to many other verbose programming languages on the market.

Coding Checklist

Attached is the coding checklist for this assignment.

Activities	Completed
Translate the design into code that will satisfy requirements.	Yes/No
Identify inadequacies in the design and requirements.	Yes/No
Document further decisions and rationale.	Yes/No
Complete? Is everything that is in the requirements and design is implemented.	Yes/No
Consistent? Ensure there are no mismatched interfaces and consistent with the design.	Yes/No
Stylistic? Exhibits good programming style.	Yes/No
Understandable? The code should be constructed in a way that is easy to read, not necessarily easy to write.	Yes/No
Modifiable? If any changes need to be applied, Modify the code as per the changes required.	Yes/No
Is the code Confirmable, Verifiable and testable? You can tell when you've met the design and requirements.	Yes/No
Is the complete code submitted to facilitator?	Yes/No
Feedback taken from facilitator.	Yes/No
Update and submit the final project code and update project related document and presentation after completion the module.	Yes/No

Submit Code

See the .txt file(s) attached in the ZIP file to access the source code.

Testing

Introduction

Testing is essential for all types of programming. It allows mistakes that otherwise would have slipped past the development team to be caught. There are many different types of testing programmers can utilize for their programs.

Testing Methods

Unit testing, which is a low level form of testing composed of test cases or a test suite written by the developer to test individual units of code.

Integration testing, is a step above individual unit testing and is the process of executing tests on a single module, groups of modules or all modules at a time pending on the needs of the program. Integration testing can be further broken down into other categories, with the two main ones being:

- **Big Bang Approach** is the approach of testing the entire set of modules as one larger unit. This requires all units to be completed before testing can execute.
- **Incremental Approach** is the approach of testing software in increments. This can be done from testing the lowest level modules upwards (bottom up) or the top most levels downwards (top-down) or a combination of both (sandwich testing).

System testing, is done after the integration of all of the modules is complete. This type of testing is typically done by professional agencies before a piece of software is to be delivered to the client. There are many types of system testing methodologies, some are as follows:

- **Load Testing** is a type of testing to determine the upper limit of users a piece of software can handle
- **Usability Testing** is a type of testing which focuses on the user's ability to utilize the application with ease along with the system's ability to meet it's objectives
- **Recovery Testing** is type of testing to see how a user recovers from crashes and to test how reliable it is for the users.
- **Migration Testing** is a type of testing to see how a piece of software can migrate from one system to another, often older infrastructures to modern based systems.
- **Regression Testing** is a type of testing to ensure changed made to the programs over time have not caused more bugs/problems with system functionality.
- **Functional Testing** is a type of testing that involves thinking of alternative/additional functions required to make improve the application
- **Hardware/Software Testing** is testing that focuses on the interactions between the hardware and software components during program execution

Acceptance Testing is testing performed by the client/end user to ensure the software meets the needs of the business and is fully functional from end to end. This can also be known as a "user acceptance testing" phase and can include end to end testing by the QA team along with user acceptance tests by the end users.

Requirements Traceability Matrix

REQ ID	NAME	DESCRIPTION	TEST CASE ID	SCENARIO	EXPECTED RESULT	ACTUAL RESULT	PASS /FAIL	COMMENTS
B-01	Create reaction collection message (host message)	Create the post to collect reactions using "!create!	TC-01	User types in "!create" in the Discord client.	Bot begins config prompt series	As expected	PASS	
B-02	Fetch reaction list from host message	Gather users who selected the reactions when "!post" is triggered	TC-02	Bot collects a list of users from a reaction	List of users that matches the users who clicked a reaction is created	As expected	PASS	
B-03	Fetch users from single reaction	When multiple reactions are present on a message, fetch users from only the reaction designated in the config.	TC-03	Authorized user issues "!pair"- bot fetches users from reactions on the host message	Fetches users only from the designated reaction	Fetches from all present reactions on a post	FAIL	Admins will need to disable new reactions by general users in the channel the bot is present in to ensure accuracy
B-03	Remove redundant reactions	Remove reactions of deleted users upon "!post" trigger	TC-04	User who clicked on reaction left guild before the command was "!pair" command was issued.	!pair command deletes reactions from users no longer in the server before pairing	As expected	PASS	
B-04	Randomize fetched list	Randomize the list of users upon "!post" trigger	TC-05	Authorized user types in the "!pair" command	Bot fetches users, then randomizes the list before printing the usernames in the embed.	As expected	PASS	
B-05	Pair users – Even Members	List the users in groups of two when user list is an even number.	TC-06	Authorized user types in the "!pair" command	Bot fetches users and prints then in groups of two	As expected	PASS	
B	Pair users – Odd Members	List the last three users as a group of three when the user list is an odd number.	TC-07	Authorized user types in the "!pair" command	Bot fetches users and prints them in groups of two, with the last	Prints users in groups of two and a single user at the bottom	FAIL	Currently assume single member is part

					group being a group of three			of the previous group of two
B-06	Post pair results	Send an embed message back to the client with the pair results	TC-08	Authorized user types in the “!pair” command	Bot sends an embed to the client with the pair title, pair description and the user pairings	As expected	PASS	
B-07	Post Alternative Message	Send an embed with the result of “No Pairings Available” when less than two users click the reaction.	TC-09	Authorized user types in the “!pair” command, host message has less than 2 real user reactions	Pair message contains "No pairings available. Try again when more users react to the host message." Instead of pair results	As expected	PASS	
B	Accept User Input for Message Customization		TC-10	Authorized user types in the “!create” command	Bot sends a series of prompts to collect user input and stores the input	As expected	PASS	
T-01	Physical host	Running machine to host bot	TC-11	Bot host executes the python file on a personal computer	Bot runs by being executed as a script on a host machine.	As expected	PASS	
T	Export to CSV	Export message ID, host title, host description, reaction, pair title and pair description to a CSV located within the pear-bot package.	TC-12	Bot finishes the prompt loop, executes the export_data() command	Content is written in the CSV file that matches the values of the answers list	As expected	PASS	
T-03	Input/Output – Creating Host Message	Ensure bot is triggered by the correct input and generates the correct output	TC-13	Authorized user issues “!create” command and finishes prompts	All input matches the user input and the embed associated with the host message sends	As expected	PASS	
T-04	Input/Output – Creating Pairing Post	Ensure bot is triggered by the correct input and generates the correct output	TC-14	Authorized user issues “!pair”	All embed output matches the user input and the embed	As expected	PASS	

				command and finishes prompts	associated with the pair message sends			
T-05	Name Format - Mention	Users in the pairing posts must be tagged properly using the @ notation	TC-15	Authorized user issues “!pair” command and more than two real users reacted to the designated reaction	Pair post includes ‘@’ handles and properly hyperlinks and pings each user tagged.	As expected	PASS	
T	Auto Add Reaction	Bot automatically adds the user specified reaction to the host message	TC-16	Authorized user issues “!create” command and finishes all five config steps	Reaction that matches the reaction chosen during config is added during when the host message is posted	As expected	PASS	
T	Config 1/5 - Prompt For Host Title	First prompt sent by the bot in the configuration series.	TC-17	Authorized user issues “!create” – configuration prompts begin to be sent by bot	Bot sends the first prompt as an embed, containing the value of prompts[0]	As expected	PASS	
T	Config 1/5 - Accept Host Title Input	First input sent by authorized user in the configuration series. Declares host message title.	TC-18	Authorized user typed in value and sent it via Discord client	Bot fetches user input, stores the value as answers[0]	As expected	PASS	
T	Config 2/5 - Prompt For Host Description	Second prompt sent by the bot in the configuration series.	TC-19	Config set 1 complete and successful – bot starts config step 2 and prompts for user input	Bot sends the second prompt as an embed, containing the value of prompts[1]	As expected	PASS	
T	Config 2/5 - Accept Host Description Input	Second input sent by authorized user in the configuration series. Declares host message description.	TC-20	Authorized user typed in value and sent it via Discord client	Bot fetches user input, stores the value as answers[1]	As expected	PASS	

T	Config 3/5 - Prompt For Reaction Choice	Third prompt sent by the bot in the configuration series.	TC-21	Config set 2 complete and successful – bot starts config step 3 and prompts for user input	Bot sends the third prompt as an embed, containing the value of prompts[2]	As expected	PASS	
T	Config 3/5 - Accept Reaction Choice Input	Third input sent by authorized user in the configuration series Declares selected reaction.	TC-22	Authorized user typed in value and sent it via Discord client	Bot fetches user input, stores the value as answers[2]	As expected	PASS	
T	Config 4/5 - Prompt For Pair Title	Fourth prompt sent by the bot in the configuration series.	TC-23	Config set 3 complete and successful – bot starts config step 4 and prompts for user input	Bot sends the fourth prompt as an embed, containing the value of prompts[3]	As expected	PASS	
T	Config 4/5 - Accept Pair TItle Input	Fourth input sent by authorized user in the configuration series. Declares title of pair message.	TC-24	Authorized user typed in value and sent it via Discord client	Bot fetches user input, stores the value as answers[3]	As expected	PASS	
T	Config 5/5 - Prompt For Pair Description	Fifth prompt sent by the bot in the configuration series.	TC-25	Config set 4 complete and successful – bot starts config step 5w and prompts for user input	Bot sends the fifth prompt as an embed, containing the value of prompts[4]	As expected	PASS	
T	Config 5/5 - Accept Pair Description Input	Fifth input sent by authorized user in the configuration series. Declares description of pair message.	TC-26	Authorized user typed in value and sent it via Discord client	Bot fetches user input, stores the value as answers[4]	As expected	PASS	
T	Export to CSV	All values fetched by the bot from the authorized user are exported to a CSV file located with the pear-bot package.	TC-27	Config loop has finished and export_data() has been initiated	CSV is created/updated containing values	As expected	PASS	

					from the answers list in the correct order			
	Import from CSV	Values from the CSV previously created by the bot are parsed and inserted into a list for further processing	TC-28	Authorized user issues "!pair" command, bot executes import_data()	The msg_list list is populated with the CSV values from the designated CSV	As expected	PASS	
	Delete Config Messages	Bot automatically deletes the configuration messages after the host message has been posted	TC-29	Host message has been sent with the corresponding reaction added	All messages after "!create" and before the host message are deleted automatically after the bot	As expected	PASS	
C-01	Discord Community Guidelines	Bot must comply with and not promote activity against Discord's Community Guidelines	TC-30	NA	Bot complies with the Discord's Community Guidelines	As expected	PASS	
C-02	Discord Developer Policy	Bot must comply with the API constraints and policy of Discord's Developer Policy	TC-31	NA	Bot complies with the Discord Developer Policy	As expected	PASS	
C-03	Discord Privacy Policy	Bot must not obtain sensitive user data without user consent as per the Discord Privacy Policy	TC-32	NA	Bot complies with the Discord Privacy Policy	As expected	PASS	
C-04	Discord Terms of Service	Bot must not violate the general Discord Terms of Services	TC-33	NA	Bot complies with the Discord Terms of Service	As expected	PASS	
C-05	Discord Store Distribution Agreement for Developers	Bot must comply with the Discord Store Distribution Agreement for Developers	TC-34	NA	Bot complies with the Discord Store Distribution Agreement for Developers	As expected	PASS	
S-01	Hide Token Data	Ensure critical token data is stored in .env files away from the code	TC-35	Source code is scanned for the Discord Token data	Bot utilizes a .env file instead of hard coded token data	As expected	PASS	
S-02	Granted Admin Role	Bot must be granted "Administration" privileges	TC-36	Administrator permission was	Bot is unable to read the commands	As expected	PASS	

		in a server Discord in order to function		removed from bot and test commands were issued by an authorized user				
S-03	Bot Usage Authorization	Bot only triggered by members who can remove members from the server.	TC-37	Unauthorized user attempts to issue the “!create” and “!pair” commands	Bot ignores user input, prints stack trace in the console	As expected	PASS	
UC-01	Create reaction host post	Bot posts embed message as a response to the command that accepts user reactions	SEE TC-01					
UC-02	Pair users/post pair message	Bot posts embed message as a response to the command that pairs users who reacted in groups of two	SEE TC-08, TC-09					
UC-03	Delete user reactions no longer on server	Delete users who are unable to be paired (due to no longer being a member of the server) before pairing/posting.	SEE TC-04					

Testing Tools

For larger projects, test management tools are available to track test cases, including but not limited to, Testpad, PractiTest, Tuskr, and Nitrate.

However, I opted out of utilizing these for the scope of this project, as this report hosts its own form of test tracking as its own matrix. The test cases are also located within the requirements traceability matrix and tracking them in several places felt redundant.

Testing Pear

Below are the testing practices I used to test Pear.

Unit Testing

Unfortunately, unit testing is not as simple as it may seem when testing Discord bots. The command methods located within bot.py are not testable using standardized unit testing without being able to accurately create a mock "context" object.

To work around this, I completely refactored my code (as mentioned in the Status report) and added unit tests for all the methods implemented in bot_functions.py. For the sake of reducing complexity, I removed the "context" parameter from set_pair_embed to run the tests, as the parameter was only used for added a cosmetic footer fetched from the Discord API and was too difficult to replicate test object wise at my knowledge level.

I utilized the PyTest API to run the tests, however, other frameworks such as Python's UnitTest, Nose2, Testify and Robot are also available for Python users to create test cases.

Test Scripts

Due to the project being small relatively and the limitations I had writing unit tests, I opted to make one since test file called test.py. This file contains all the unit testing methods that were executed for this assignment.

```
from pear_bot import bot_functions
import pandas as pd

test_prompts = [
    "prompt 1",
    "prompt 2",
    "prompt 3",
    "prompt 4",
    "prompt 5"
]
test_answers = [
    "test title 1",
    "test desc 1",
    "☑",
    "test title 2",
    "test desc 2"
]

test_msg_list = [
```

```
    123454321,
    "test title 1",
    "test desc 1",
    "☑",
    "test title 2",
    "test desc 2"
]


class TestMessage:
    id = 987654321


class TestUser:
    name = ""
    mention = ""


test_users = [TestUser() for i in range(8)]
for i, user in enumerate(test_users):
    user.id = i * 997654
    user.name = "user" + str(i)
    user.mention = ""


def test_export():
    print()
    bot_functions.export_data(TestMessage, test_answers)
    contents = pd.read_csv(r'root_message.csv')
    assert contents is not None


def test_import():
    print()
    contents = bot_functions.import_data()
    assert contents is not None


def test_create_prompt_embed():
    for i, prompt in enumerate(test_prompts):
        embed = bot_functions.create_prompt_embed(i, test_prompts,
test_answers)
        if i == 0:
            assert embed.title == test_answers[0]


def test_set_pair_embed():
    print(" >> TESTED WITH USER LIST LENGTH OF " + str(len(test_users)))
    embed = bot_functions.set_pair_embed(test_users, test_msg_list)
    if len(test_users) > 2:
        assert embed.title == test_msg_list[4]
        assert embed.description is not None


def test_config_fields():
    for i, prompt in enumerate(test_prompts):
```

```
        embed = bot_functions.create_prompt_embed(i, test_prompts,
test_answers)
        #  config = bot_functions.config_fields(i, embed, test_answers)
        if i == 0:
            assert embed.title == test_answers[0]
        elif i == 1 or i == 2 or i == 3:
            assert embed.title == test_answers[0]
            assert embed.fields.count(1) is not None
        else:
            assert embed.title == test_answers[0]
            assert embed.fields.count(2) is not None
```

Test Output

Below is the output of test.py:

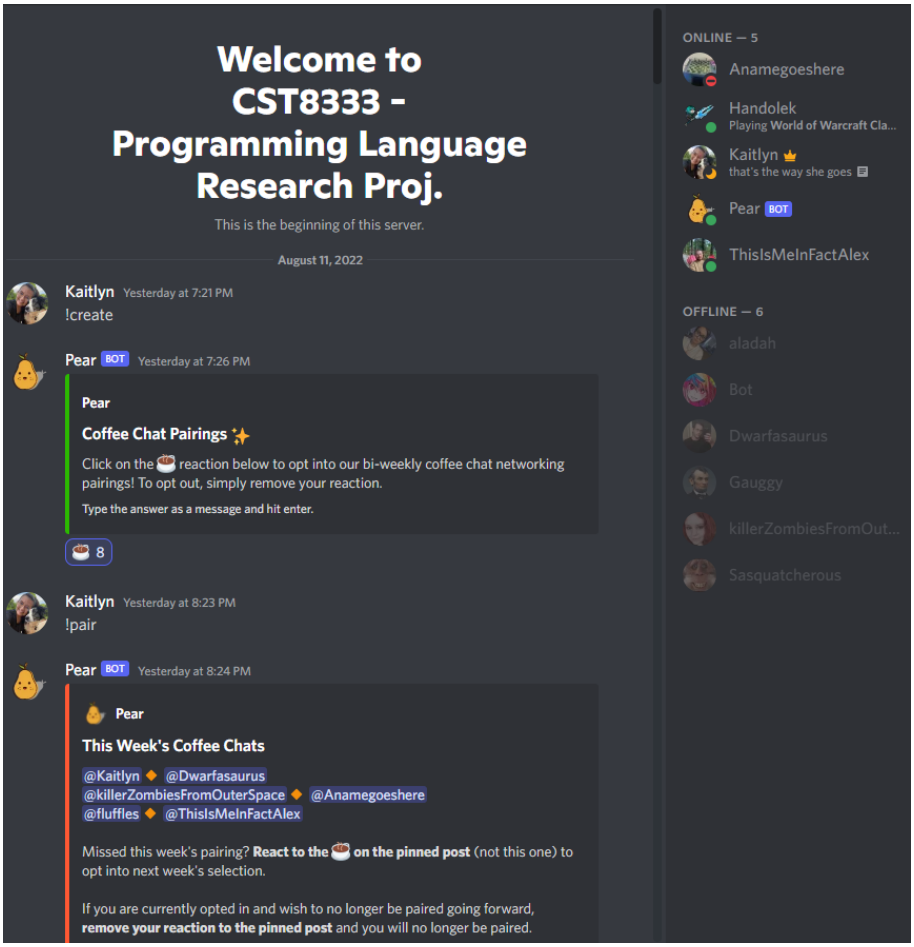
```
Testing started at 7:22 PM ...
Launching pytest with arguments
C:/Users/Katie/PycharmProjects/Pear/pear_bot/test.py --no-header --no-summary
-q in C:\Users\Katie\PycharmProjects\Pear\pear_bot

============================= test session starts
=============================
collecting ... collected 5 items

test.py::test_export PASSED                                              [
20%]

test.py::test_import PASSED                                              [
40%]

test.py::test_create_prompt_embed PASSED                                 [
60%]
test.py::test_set_pair_embed PASSED                                      [
80%] >> TESTED WITH USER LIST LENGTH OF 8

test.py::test_config_fields PASSED
[100%]

============================== 5 passed in 0.53s
==============================

Process finished with exit code 0
```

Integration Testing

I manually tested each command and their respective methods as a group of modules as new features were added in. This began by testing the “!pair” command on a static message with pre-loaded reactions to ensure the bot was triggered accurately. This grew as features were added, each command was tested as its own unit.

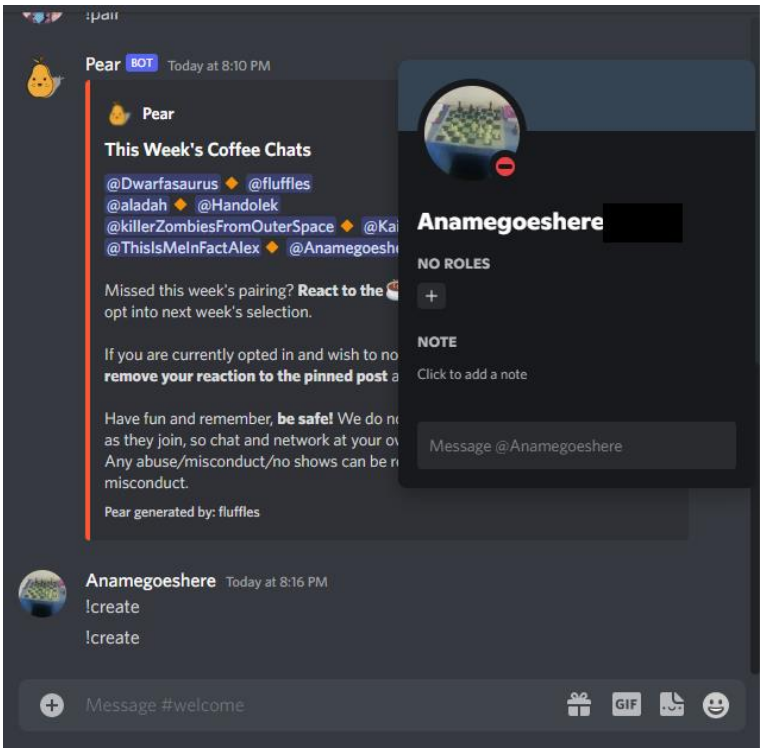
Acceptance Testing Phase

To preface this section, I would like to thank all my friends and cohorts who helped me test this bot. I could not have done manual testing to the level I was able to achieve without your participation in clicking reaction buttons in a random Discord server for me.

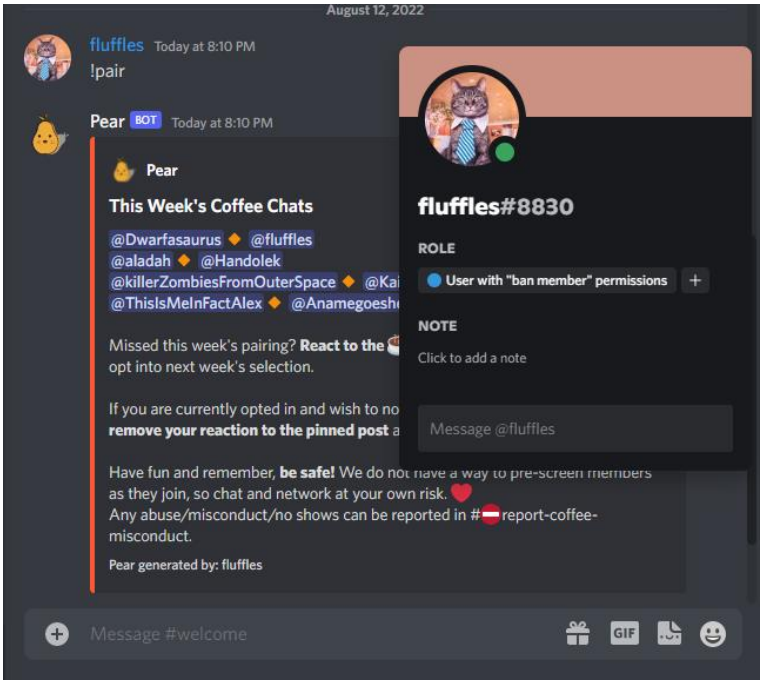


After both commands have executed end to end without errors, with the data being stored in the CSV correctly being written and read, I moved onto a testing phase like acceptance testing. I spoke with the users in the test server I created and ensured they were able to read and understand the commands, configuration process and output as if they were the authenticated user.

For example, below is a user who isn't authenticated trying to execute the bot commands:



And below is a user who is authorized (with ban permissions) issuing the command:



Final Test Report

Pear passed 94.6% (35/37) of its test cases present in the test matrix, and 100% of the unit test cases executed in test.py.

Pear failed a total of 2 test cases out of 37, TC-03 and TC-07, with a prioritization label of “medium”. Neither test case failed are functional requirements but should be prioritized to be fixed within the next release.

Deployment

Introduction

Deploying Pear to a personal server can be done in minutes. The code is hosted on GitHub for anyone to use and can be executed by anyone with a personal computer, Discord account and access to a server with Administrator privileges.

Software Integration Plan

Introduction

Pear is an open-source project fueled by my passion for networking and automation. I have designed this software to be used for any scenario that would require pairs on the Discord platform.

Pear Requirements

Pear is designed to only function on the Discord platform, a social network consisting of servers where users gather to chat, network, play games and more.

This bot requires discord.py, a Python API for Discord. Should the API at any point no longer be available, the bot will need to be rewritten using an API that is.

Pear requires a user to have a Discord account and have Administrator permissions to exist in a Discord. To execute a command using Pair, the user must have the “ban users” permissions designated to them via a role.

Since Pear is not hosted via a cloud server like many live bots are, it will need to be downloaded and recreated as a personal bot via the Discord developer portal. Instructions on how to do this can be found here.

After the bot creation, the user will need to enter the TOKEN data generated in the .env file provided in directory. This will replicate the Pear logic as your own bot.

For Pear to function, the bot.py file *must* be running on your personal machine while commands are issued. This can be done by either executing the bot.py as a Python script, or by opening toe repository in an IDE and executing from there.

Conclusion

Learning something new is always a memorable experience but doing so while doing something you love makes it beyond enjoyable. This project gave me the opportunity to approach a solution to a real-world problem I faced at an angle I wouldn't have otherwise. I am thankful for the things I have learned this semester and look forward to seeing what's next for Pear.

Thank you,

Kaitlyn

Questions?

Should any questions arise, feel free to contact me via email at gati0010@algonquinlive.com.

References

Algonquin College. (2020a). *CST8333 Budget Template*. Algonquin College.

Algonquin College. (2020b). *CST8333 Timeline Template*. Algonquin College.

Algonquin College. (2020c). *CST8333 Assignment 1 Outline*. Algonquin College.

Algonquin College. (2020c). *CST8333 Assignment 2 Outline*. Algonquin College.

Algonquin College. (2020c). *CST8333 Assignment 3 Outline*. Algonquin College.

Chi, E. (2020, August 10). *How to Make Discord Bot Commands in Python*. Better Programming; Medium. https://betterprogramming.pub/how-to-make-discord-bot-commands-in-python-2cae39cbfd55

Discord. (n.d.-a). *Discord Developer Portal — API Docs for Bots and Developers*. Discord Developer Portal. Retrieved June 8, 2022, from https://discord.com/developers/docs/intro

Discord. (n.d.-b). *Discord Store Distribution Agreement for Developers (Self-Service)*. Discord Developer Portal; Discord. Retrieved May 31, 2022, from https://discord.com/developers/docs/policies-and-agreements/store-distribution-agreement

Discord. (n.d.-c). *OAuth2*. Discord Developer Portal; Discord. Retrieved June 8, 2022, from https://discord.com/developers/docs/topics/oauth2

Discord. (n.d.-d). *Permissions*. Discord Developer Portal; Discord. Retrieved June 10, 2022, from https://discord.com/developers/docs/topics/permissions

Discord. (2020, July 1). *Discord Developer Policy*. Discord Developer Portal; Discord. https://discord.com/developers/docs/policies-and-agreements/developer-policy

Discord. (2022a, February 25). *Community Guidelines*. Discord. https://discord.com/guidelines

Discord. (2022b, February 25). *Privacy Policy*. Discord. https://discord.com/privacy

Discord. (2022c, February 25). *Terms of Service*. Discord. https://discord.com/terms

Duke, R. (2022). *What is a Work Breakdown Structure*. Workbreakdownstructure.com. https://www.workbreakdownstructure.com/

Hamilton, T. (2022, July 2). What is Test Driven Development (TDD)? tutorial with example. Guru99. Retrieved August 6, 2022, from https://www.guru99.com/test-driven-development.html

Hamilton, T. (2022, June 18). Integration testing: What is, types, top down & bottom up example. Guru99. Retrieved August 8, 2022, from https://www.guru99.com/integration-testing.html

Hamilton, T. (2022, June 25). Different types of testing in software: 100 examples. Guru99. Retrieved August 10, 2022, from https://www.guru99.com/types-of-software-testing.html

MiiaHash. (2021, May 3). Microservices architecture (discord bot). *MCMARKET*. https://www.mc-market.org/threads/662952/#post-4826533

Pheonix, J. (2022, May 12). The Complete List Of Python Assert Statements. Just Understanding Data. Retrieved August 10, 2022, from https://understandingdata.com/list-of-python-assert-statements-for-unit-tests/

Pngtree. (2021). meb style yellow cute pear clipart [Online]. In 阿Go (Ed.), *Pngtree*. https://pngtree.com/freepng/meb-style-yellow-cute-pear-clipart_5867760.html

Rapptz, D. (n.d.). *Cogs*. Discord.py. Retrieved June 9, 2022, from https://discordpy.readthedocs.io/en/stable/ext/commands/cogs.html

Rapptz, D. (2019, September 30). *Welcome to discord.py*. Discord.py. https://discordpy.readthedocs.io/en/stable/

Rapptz, D. (2022). *A primer to gateway intents*. A Primer to Gateway Intents. Retrieved July 19, 2022, from https://discordpy.readthedocs.io/en/latest/intents.html#privileged-intents

Real Python. (2022, April 14). Getting started with testing in Python. Real Python. Retrieved August 9, 2022, from https://realpython.com/python-testing/

Seewald, D. (2022, April 11). *Send an Embed with a Discord Bot in Python*. Python in Plain English; Medium. https://python.plainenglish.io/send-an-embed-with-a-discord-bot-in-python-61d34c711046

Taneja, N. (2021, August 27). *Architecting discord bot the right way*. DEV Community. https://dev.to/itsnikhil/architecting-discord-bot-the-right-way-383e

The Python Software Foundation. (2022). *What is python? executive summary*. Python.org. Retrieved July 14, 2022, from https://www.python.org/doc/essays/blurb/

Warren, T. (2021, November 17). *Discord is quietly building an app empire of bots*. The Verge. https://www.theverge.com/2021/11/17/22787018/discord-bots-app-discovery-platform

What is system testing? types & definition with example. Guru99. (2022, July 2). Retrieved August 8, 2022, from https://www.guru99.com/system-testing.html

Appendix A: Module 10 Checklist

Activity	Completed
Completion of Tools & Technology phase.	Yes/No Done in Assignment 2
Completion of Project Timeline phase by deciding time frame for project development.	Yes/No
Completion of Source Code Implementation phase.	Yes/No
Completion of Project Progress Report phase.	Yes/No
Submission of Power Point Presentation includes details of Tools & Technology, Project Timeline and Source Code Implementation of your project.	Yes/No Done in Assignment 2
Submission of Project progress Report [word file], having detailed information about phases from project Requirement Analysis to Source Code Implementation.	Yes/No
Feedback taken from facilitator regarding all your submissions [presentation and report].	Yes/No
Discussion between Facilitator and Developer regarding any updates suggested by the facilitator.	Yes/No Feedback from Assignment 2
Submit the final presentation and final project report to the facilitator within the given deadline.	Yes/No One day late
Facilitator approved the final submission-2 and allows you to start next phase of your development.	Yes/No

Appendix B: Module 11 Checklist

Activity	Completed
Review/refine or create the Test Plan.	Yes/No Conducted continuous manual testing from start to finish
Review the results of the system verifications.	Yes/No

Determine test tracking tools, defects tracking list, and prepare defect tracking log.	Yes/No
Refine the data and load data into the test environment.	Yes/No
Execute test scripts and create a test report.	Yes/No
Report results of test scripts and update test issues and defect tracking list.	Yes/No
Store the data collected during testing in accordance with configuration management procedures.	Yes/No No management procedures for this project
Validate the system with functional and structural testing.	Yes/No
Evaluate test reports to determine accomplishments and report status. Update the test log.	Yes/No
Develop Final Test document summarizing all tests, results, defects found, etc.	Yes/No
Plan and conduct a test review. Gather feedback and approval for the testing from facilitator.	Yes/No Did not submit to facilitator
Implement "build" procedures, execute smoke tests and fix any errors that prohibit the build from being tested.	Yes/No
Develop test scripts based on requirements, design specifications, and code development.	Yes/No
Create the test environment and setup automated test procedures, as appropriate. Adapt test reporting procedures based on size, complexity, and specific project needs. Develop test data management strategy.	Yes/No
Review the System/Application Support Checklist and update support documents, as appropriate.	Yes/No

Appendix C: Module 12 Checklist

Activity	Completed

Review/revise software integration process.	Yes/No
Define development strategy to build application.	Yes/No
Evaluate resources for the Development Phase.	Yes/No
Review code to ensure it meets specifications.	Yes/No
Prepare for and conduct final development approval.	Yes/No
Has the website design been viewed and approved by the facilitator?	Yes/No
Is project/website defect free?	Yes/No The two defects/failed test are not essential to running the program
Have planned data creation/conversion activities been executed or are they on schedule to be completed as planned?	Yes/No
Are activities to enable the operation and maintenance of the product on schedule and ready for the planned completion?	Yes/No
Have environment preparation activities (e.g. correct OS, memory etc.) been completed?	Yes/No On a smaller scale (Discord/host machine)
Is the code working properly as a workable application?	Yes/No
Submit code and working application screenshots to facilitator.	Yes/No
Feedback taken from facilitator about submitted code and application.	Yes/No

Correct code according to updates given by facilitator.	Yes/No
Submit final code and workable application to the facilitator.	Yes/No
Add programming code and screenshots of each module of workable application in project Report.	Yes/No

Appendix D: Module 13 Checklist

Activity	Completed
Select proper template [should be simple] for your project presentation.	Yes/No
Your presentation should be separated in three parts: Introduction, Details and Conclusion.	Yes/No
Is introduction added to the presentation?	Yes/No
Is project requirement collection information added to the presentation?	Yes/No
Is feasibility report information added to the presentation?	Yes/No Done in Assignment 1
Is project design information added to the presentation?	Yes/No Done in Assignment 1
Graphics, Images, Flowcharts etc. related to your project added to the presentation?	Yes/No
Are tools and technology related to the project added to the presentation?	Yes/No Done in Assignment 2
Is the project timeline information added to the presentation?	Yes/No

Is information regarding project code [You can include a hyperlink in your presentation which opens your code] such as environment and IDE used in project added in the presentation?	Yes/No
Are workable application screenshots added in the presentation?	Yes/No
Are advantages of your project added in the presentation?	Yes/No
Is future possible work for your project added in the presentation?	Yes/No
Is the conclusion added to the presentation?	Yes/No
Is your presentation submitted to the facilitator?	Yes/No
Feedback taken from the facilitator?	Yes/No
Is the final presentation submitted to the facilitator within the given time line?	Yes/No One day late

Appendix E: Assignment Grading Rubric

Criteria	**Excellent 80-100%**	**Good 50-79%**	**Requires Improvement <50%**	**Points**
Assignment Quality	All information offered is accurate All views are clearly expressed and well explained Contains original ideas, connections, or applications	Most information offered is accurate Most views are clearly expressed and explained Contains mainly original ideas, connections, or applications	Some or no accurate information offered Views are rarely or never clear and require further explanation Many non- original ideas, or unclear connections or applications	**/15**
Assignment Knowledge and Skills Demonstration	Clear, concise synthesis of course content to demonstrate understanding of topic	Evidence of some synthesis of course content to demonstrate understanding of topic	Lack of evidence or weakness in the synthesis of course content to demonstrate	**/20**

Criteria	Excellent 80-100%	Good 50-79%	Requires Improvement <50%	Points
	All ideas are clearly developed, organized logically, and connected with effective transitions Explores ideas, supports points fully using a balance of evidence, and uses effective reasoning to make useful distinctions All relevant course and topic links are made	Some unified and coherent ideas are developed with effective transitions Supports most ideas with effective examples, and/or references, and details, makes key distinctions Most relevant course and topic links are made	understanding of topic Develops and organizes ideas that are not necessarily connected. Some ideas seem illogical and/or unrelated Presents ideas in general terms; most ideas are inconsistent/unsupported, and reasoning is flawed or unclear Some or no relevant course and topic links are made	
Assignment Structure	Formatted as per assignment details Structure and format enhance delivery of the information Clear language is used which leads to easy readability Correct grammar and spelling are consistently used	Formatted as per assignment details in most components Structure and format fits well with the delivery of the information Mostly clear language is used with minor readability issues Few or no spelling and/or grammatical errors	Formatting has not been followed Structure and format are unclear and impedes delivery of the information Language used is often unclear which impedes readability Many spelling and grammatical errors	**/5**
Total Points				**/40**