Capstone Project Report

MineTicket Project

Kennesaw State University

IT 4983-01/W02 Capstone Spring 2024

Web Link: https://mineticket.weebly.com/

Group 2

Palmer Brown

Arturo Martinez

DJ Rice

Grace Shell

Solomon Thao

Last Updated on 4/21/2024

Executive Summary

Introduction

This document seeks to inform our stakeholders (and related associates) of details pertaining to the MineTicket project, as well as the intricacies of the project's implementation and design. We would like to express our thanks to the KSU eSports club management as well as our capstone professors and coordinators for their participation and effort towards seeing to the completion of this project. Our group (which will henceforth be referred to as "Group 2") is confident in its ability to design and implement a system that will suit the needs of the KSU eSports club and community.

Stakeholder Needs

To Group 2's understanding, the needs of the main stakeholders (KSU eSports) are as follows:

- **System Development**
  - The development of a system that links a Discord ticketing bot to the KSU eSports Minecraft server, ideally through a database containing ticket information
- **Convenience and Longevity**
  - The aforementioned system must be designed in such a way as to promote efficiency and convenience for KSU eSports moderation staff.
    - Specifically on the side of Minecraft, there is a preference from the stakeholders to integrate the system with the in-game chat.
  - The aforementioned system must be designed in such a way as to prolong its supported lifespan after the project's completion. Versatility across multiple versions of Minecraft and Discord is therefore preferred.

Stakeholder Outcomes

Group 2 understands that the project's stakeholders wish for the final product to result in a more efficient way for their staff to handle tickets from the KSU eSports community, which will assist in quicker response times to user issues, and a better overall experience for both users and moderation staff. A system that is user-friendly from all angles (in terms of moderation staff, server users, administration staff, etc.) is desired.

Table of Contents

Background

The KSU eSports management team runs into the same issue that any organizational moderation team is liable to encountering: the issue of maintaining a long-standing and low-maintenance system in which complaints, issues, and concerns can be efficiently communicated and acted upon by an appropriate member of staff. In the particular case of KSU eSports, the environment of a Minecraft server (especially one that is open to the public according to KSU eSports moderator Derek Comella, also known by his username “qartho”) is prone to many different issues that may need immediate attention from moderation staff. Such incidents include hacking, “griefing”, bug exploitation, and violations of community rules set by the KSU eSports management team.

Group 2 seeks to resolve these issues by offering solutions that will align with the needs of the stakeholder. As previously mentioned in the project introduction, Group 2 seeks to create solutions that are user-friendly, intended for long-term use, and low-maintenance in terms of upkeep.

Project Scope

Goals	• Create a Discord bot with the following functionalities:
	  ○ Chat-based interaction
	  ○ Text channel creation
	  ○ Button interaction
	• Create ticket database functionality within a Minecraft environment
	• Create a database as a "middleman" system in which data is transferred from Minecraft to Discord through the database
	• Implement ticket functionality across the Discord/Minecraft applications

Deliverables	• Discord bot • Ticketing database • Demonstration of implemented features (e.g. cross-application functionality)
Resource Constraints	• Because this is a project with no funding, Group 2 is severely limited in terms of monetary cost • Because Group 2's members are all students, there may also be issues regarding time cost depending on the student • Regardless, because this project is part of a capstone project and is not particularly cost-intensive, the lack of resources should not be a major issue • Graciously, Group 2 has also received multiple offers for KSU eSports resources to use during the testing and implementation phases of the MineTicket project

Technical Background

For the MineTicket project, an understanding of Minecraft, Discord, Python, and MariaDB is needed to grasp the basic intricacies of how the project is currently intended to work. A list of helpful concepts to understand is as follows:

- Minecraft
    - Different Minecraft versions and editions
    - Minecraft chat command functionality
    - A general understanding of Minecraft multiplayer communities
    - Minecraft API
- Discord
    - Bot chat command functionality
    - A general understanding of Discord's user-facing structure, specifically within text channels and chat command functionality
    - Discord code methodology
    - Discord API
- MariaDB
    - Navigation of SQL environments
        - SQL language
    - Functionality with Python
        - Pulling data from a database
        - Connecting a database to a Python environment
- Python
    - Group 2 mainly utilized Visual Studio Code to create the MineTicket bot using Python

Further information and cited research on these topics can be found in the "Technical Summary" sub-section of the "Project Outcomes" section.

Project Outcomes

Assessment of Project Outcomes

Overall, the project was a major success. The following is a list of goals and deliverables, along with general commentary on each. ***Commentary will be italicized and bolded.***

- Goals
  - Create a Discord bot with the following functionalities:
    - Chat-based interaction
    - Text channel creation
    - Button interaction
      - ***All mentioned functionalities were implemented by the project's due date.***
  - Create ticket database functionality within a Minecraft environment
    - ***MariaDB connection with the Python code allows for easy navigation of a database structure. The database itself is set up to obtain data from a Minecraft environment.***
  - Create a database as a "middleman" system in which data is transferred from Minecraft to Discord through the database
    - ***MariaDB functionality with Python allows for the database to communicate with both Discord and Minecraft simultaneously.***
  - Implement ticket functionality across the Discord/Minecraft applications
    - ***Ticket functionality implemented by the project's due date.***

- Deliverables
  - Discord bot
    - ***Complete, planned to be delivered by due date***
  - Ticketing database
    - ***Complete with instructions on MariaDB setup and connection***
  - Demonstration of implemented features (e.g. cross-application functionality)
    - ***Features were regularly displayed at sponsor/milestone meetings as well as upon request from the project sponsor.***

Technical Summary – Milestone 1

Much of Milestone 1's work has been dedicated to connecting a MariaDB database to the MineTicket Discord bot which runs on Python, as well as implementing the base functionality of the bot. The consultation of several resources on the MariaDB site was crucial to understanding the process of proper implementation. For MariaDB Community Server version 10.6, database

configuration and testing were handled by issuing commands through the terminal (1). Connecting the database to the developed Python-based Discord bot required the MariaDB-Python Connector software, allowing Group 2 to “use and administer databases from within [the] Python application” (2). The combination of the MariaDB server and the Python connector software allowed the connection between the Discord bot and the database to function properly. Tests were completed in Group 2’s own test Discord server which showed that the bot could successfully pull and display data from an established database.

One problem tackled during this Milestone was that of SQL injection. Since the MineTicket bot is meant to take in user input to be used in a SQL query, the bot runs the risk of a malicious input which can execute unintended queries. To briefly explain SQL injections, a malicious user can input either an escape sequence of characters, such as a quote or a well-placed backslash, and then input a query they wish to execute on the server, such as SELECT * FROM passwords, or DROP DATABASE database. For the project's use case, the likelihood of these attacks is very slim, but still a potential threat. As such, Group 2 created an SQL sanitization module, which handles construction and execution of SQL queries in a manner that eliminates the risk of these attacks being a possibility.

In order for the Python-based bot itself to function properly within Discord, Group 2 required the usage of discord.py, “a library for Python to aid in creating applications that [utilize] the Discord API” (3), thereby allowing for connectivity between the bot and Discord. Especially important to bot commands and basic in-app functionality was the concept of “intents”, which “allows a bot to subscribe to specific buckets of events” (4). In other words, Discord intents are a form of coded-in permissions for a bot to use, which is primarily helpful in ensuring a more securely coded bot that does not have unnecessary permissions or abilities (which would be particularly harmful in the case of a bot security breach).

Other concepts within the space of Python have also instrumental in ensuring Group 2’s code remains readable and efficient, which reinforces KSU eSports’s desire for low-maintenance structures that are easily understandable which can be picked up by others if needed in the future. For example, discord.py’s cog organization allows for “[organizing] a collection of commands, listeners, and some state into one class” (5), which assists in breaking down command lists to more readable portions. Additional efficiency-related concepts in Python include the return value annotation and “args/kwargs”. Return value annotation is “a way to document [the] code elegantly in-line, by allowing [Group 2 and other administrative users] to simply describe the data type of the ‘thing’ the function returns” (6, 2020), which makes returned function data readable and understandable to others. Args (non-keyword arguments) and kwargs (keyword

arguments), which allow the passing of variable-length groups of arguments into Python functions, increase flexibility within the code (7, 2023).

Technical Summary – Milestone 2

The main goal of Milestone 2 was to have the MineTicket bot functionality implemented at least at a base level. Group 2 desired to have a demonstrable prototype of the MineTicket bot that could be shown at the meeting for Milestone 2 with KSU eSports.

One of the main tenets of bot functionality was the usage of commands from within the Discord environment for users to interact with the bot. Ensuring that command implementation was done seamlessly, efficiently, and effectively were paramount to proper function of the bot. There are numerous examples of how discord.py documentation was used, but one of the main usages of the documentation was to properly implement hybrid commands, which is a method of invoking a command "... as both a text and a slash command. This allows [programmers] to define a command as both slash and text command without writing separate code for both counterparts" (9). Having the ability to simplify the project code helps readability from both a developer end and from the end of future developers that may attempt to edit the code to fit future standards. Synchronization of the command list was found to be best handled through its own dedicated "sync" command (10) which can be used at any point when needed within the Discord environment.

In relation to the MineTicket bot's command functionality, both Group 2 and KSU eSports agreed that button functionality was paramount to the accessibility of the bot to be used by developers, administrative staff, and regular server users without any special permissions. To properly implement buttons, Group 2 required discord.py's button UI documentation (11) and the development of the command list. Through these developments, proper button functionality is showing significant progress, but still may need some testing before fully operational at an acceptable level.

Another focus of the Milestone 2 period was the focus on getting data to and from a SQL database through the usage of the cursor class within MariaDB's import selection. The SQL-side of the MineTicket project was able to be navigated properly by the bot through the method of creating a connection between Python and the database, having the cursor work as intended depending on the functionality needed (ex. inserting or pulling data), and then closing the connection once completed before returning any required data, if applicable. Helpful cursor methods included "cursor.commit" which allowed Python to push changes towards the database, and "cursor.close" which closes the cursor until needed again. The "cursor.execute" method in

particular was helpful to database functionality due to its ability to allow for the preparation and execution of SQL statements (12).

Furthermore, there were significant technical difficulties regarding Wix hosting the MineTicket project site due to significant downtime and issues found when attempting to access the Wix site. However, a quick fix was found by instead having a different site hosted by Weebly, which has proven to be more stable and accessible.

Technical Summary – Milestone 3

Milestone 3 presented the issue of button persistence, as currently any buttons sent by the bot would cease to function should the bot be restarted, or if connection to the server was lost. With this problem solved, the MineTicket bot would be considered a finished project in Group 2's eyes, and ready for sponsor use in the field.

With each message sent in Discord, there are certain properties which remain hidden from the user. One of these properties is a message's "view". This view is a way for these messages to present UI elements to users, such as buttons in this case (11). The problem is that, when the MineTicket bot is restarted either due to a necessary hardware reboot, software update, or internet outage, the bot effectively "loses track" of these views and is unsure of what to do with interactions received from them. The solution to this problem is a functionality known as persistence.

Persistence allows these views to be re-accessed following a reboot, meaning that interactions sent from them are not lost upon reset. However, a problem with persistent views is that each instance of a view must be made persistent individually, and this is performed when the bot initially boots up. With the MineTicket bot's implementation, this would not work. Each button contains a custom ID where the associated staff ticket ID is stored, meaning each button is unique to each message. If the bot were to make buttons persistent on a boot instance, it would either take an increasingly long time for the bot to reboot, or it would completely disregard inputs from certain buttons. Thus, the introduction of dynamic item classes was needed to further ensure product stability.

Dynamic items appear to be the same as regular Discord UI items, such as buttons. However, there are several key differences that make them extremely useful. First, class declaration of a dynamic item can be declared persistent, meaning that any dynamic item of that class created by the bot is automatically persistent, allowing the bot to not only properly receive interactions post reboot, but retain the information stored within the custom ID. The second key difference is that dynamic items have special functionality for pulling information from the custom ID via regex match. This makes it extremely easy to create the button when needed, store information in the custom ID, and then use that information to perform different actions or pass

along to helper functions. This, combined with the ability to overwrite the callback function of the dynamic item (much like with a regular button) allows dynamic items to be incredibly powerful, and they were exactly the solution needed for the MineTicket bot.

```python
class DynamicButton(discord.ui.DynamicItem[discord.ui.Button], template=r'button:(?P<type>[a-zA-Z]+):(?P<id>[0-9]+)'):
    def __init__(self, *, ticket_id, button_type:str, button_style:discord.ButtonStyle=discord.ButtonStyle.green) -> None:...

    @classmethod
    async def from_custom_id(cls, interaction: discord.Interaction, item: discord.ui.Button, match: re.Match[str], /):...

    async def callback(self, interaction: discord.Interaction) -> None:
        #TODO:
        # implement add button
        if self.button_type == "claim":
            await claim_ticket_helper(interaction, self.id)
        elif self.button_type == "channel":
            await create_channel_helper(interaction, self.id)
        elif self.button_type == "close":
            await close_ticket_helper(interaction, self.id)
        elif self.button_type == "open":
            modal = TicketModal()
            await interaction.response.send_modal(modal)
        else:
            embed = discord.Embed(
                title = "Unexpected Error",
                description = f"An unexpected error has occured, and the button you have clicked is not a valid button. Please report this issue.\nButton Type: {self.button_type}",
                color=discord.Color.red()
            )
            await interaction.response.send_message(embed=embed)
            return
```

Dynamic Buttons Class from bot_manager.py

Besides the dynamic items changes and persistence updates, the MineTicket bot also received the ability to parse JSON strings sent within a specific channel. By using Discord as an intermediary and properly restricting the “feed” channel’s permissions to prevent improper messages, the ability to parse these JSON strings gives the bot the ability to be connected to almost any other program running anywhere else in the world, provided that the bot has the permissions to send messages within this channel.

Each JSON string contains a variety of information about what the bot needs to do, centered around the “event” variable of the JSON string. The three primary events are “create”, “claim”, and “close”. Just like the buttons and commands within Discord, any JSON string sent using one of these three commands can interact with the database and create/update existing tickets, while also keeping staff members in the loop and not requiring any special interactions. Tickets appear in the same channel they normally would, staff members are still able to claim tickets from within discord even if they originate from a different source, and tickets created within Discord can also be managed from other sources, such as a Minecraft server staff member. This flexibility was crucial to the sponsor, and so implementing it properly within MineTicket was a top priority.

Project Planning

The MineTicket project would not be in the position it is now without the management practices we implemented in order to reach our desired milestone progression. This section will provide certain insights into how we progressed under our management methods.

Overview

As of Milestone 1, Group 2 operates at a functionally acceptable capacity. Weekly meetings and discussions within Microsoft Teams keep each member updated on progress and accountable for their portions of the required work.

The seventh volume of *A Guide to the Project Management Body of Knowledge* contains the Tuckman Ladder, a development model regarding the stages of team development (8, 2021). The five stages of the Tuckman Ladder are: forming (initially coming together), storming (internally competing and identifying with roles and positions within the team), norming (beginning to function as a team with understood roles), performing (operating efficiently and synergistically), and adjourning (completing the project and possibly dispersing to work on others). While there is no rubric for Group 2 to be graded on in terms of where it aligns itself on this ladder, it's safe to say that Group 2 is beginning to function as a team that understands the roles of each team member. Therefore, while improvements need to be made regarding performance in a synergistic and highly efficient manner, Group 2 is currently in the "norming" stage and are likely trending towards the next.

As of Milestone 2, Group 2 has improved on their previous challenges to optimal efficiency. As predicted in the first draft of this document, Group 2 has entered the "performing" stage, where each member has a better idea of their role in the group's progress and communication between different branches of work is openly communicated. Synergy between group members, whether collaboratively or individually, has reached an all-time high within the project's timeline.

Milestone 3 solidified the working order and task assignments of all team members. By this point, Group 2 transitioned from the "performing" stage and is now in the "adjourning" stage. Each member has found comfortable individual positions within the dynamics of the group, and are ready to continue post-project processes and full adjournment. Group members have expressed positivity and fulfillment at the completion of the project and may collaborate in the future.

Process

Milestone 1's most important aspects were the implementation of the basic Discord bot functionality and the MariaDB database. Group 2 managed to achieve this through separating task responsibilities by familiarity—some members were better versed in Python and Discord, while others were more used to database structures. Since this was the first major milestone for this project, the greatest challenge was understanding how Group 2 worked as a team and what each member's strengths and weaknesses were.

- Goals reached:
    - Basic Discord bot functionality
        - Bot setup, basic responses to inputs, etc.
    - MariaDB database implementation
    - Group task delegation standards

Milestone 2's most important aspects were the functionality of the bot within the Discord environment, mainly consisting of code that implemented commands and button functionality, as well as project website creation and combining the Discord bot functionality with the project's SQL database, along with the actual navigation of said database through the use of various commands and JSON methods. As with Milestone 1, each group member's familiarity with certain aspects of the deliverables was considered when assigning work.

- Goals reached:
    - Implementing button functionality
        - Basic button interactions and testing
    - JSON configuration for database interaction

Milestone 3's most important aspects concerned button persistence and developing both JSON parsing methods and the documentation of the project through the GitHub Wiki and the project site. Persistence, as stated before, proved to be a significant roadblock to project completion and required significant attention from all group members. Regardless, the persistence issue was solved and the finalization of the project came shortly afterwards.

- Goals reached
    - Button refinement and finalization
        - Persistence update
    - JSON updates

        - Backend finalization
        - Flexibility
    - Code cleanup
        - Removal of debug commands meant for development use
    - Documentation
        - GitHub Wiki
        - Final website update

Team Contribution Summary

Milestone 1

Palmer Brown	Arturo Martinez	DJ Rice	Grace Shell	Solomon Thao
Codebase guidance Code implementation for MariaDB to Python data transfer	Implementation of various Discord bot chat functions Creation of Discord test bot*	MariaDB research and implementation SQL injection sanitization	Implementation of various Discord bot chat functions Creation of Discord test bot*	MariaDB to Python connectivity research Creation of Discord test bot*

Creation of Discord test bot*		Bot functionality with MariaDB		

***=** Denotes creation of a personal Discord test bot for the member, each of which was referenced and used to create a final MineTicket test bot

Milestone 2

Palmer Brown	Arturo Martinez	DJ Rice	Grace Shell	Solomon Thao
Interactive button testing and implementation Integration of SQL commands to commands Tested out various UI designs for ticket forms	Implemented bot commands and connected to MariaDB Create, Claim, Close commands drafted and worked on Drafted future possible commands	Continued MariaDB Schema testing Reworked bot interaction framework Create, Claim, Close command work Testing UI display for players	Implemented help command Website editing	Interactive button testing and implementation Various note-taking and short-term planning duties Research paper drafting

Milestone 3

Palmer Brown	Arturo Martinez	DJ Rice	Grace Shell	Solomon Thao
JSON Parsing backend functionality Robust button backend based on button context ReadTheDocs testing, eventual swap to GitHub Wiki instead Filling out documentation for overall codebase	JSON Research JSON Parsing backend functions Attempting to solve Persistence issues with team members Code review and cleanup	Finalized MariaDB schema Implemented button persistence Created streamlined setup process Re-implemented UI display for ticket creation	ReadTheDocs attempt The GitHub Wiki Website editing	Button testing/code functionality Research paper drafting Various note-taking and short-term planning duties Code cleanup and embed implementation across code Persistence research

and bot functionality Code cleanup and refactoring across entire codebase				

Workload Summary

Milestone 1

Palmer Brown	Arturo Martinez	DJ Rice	Grace Shell	Solomon Thao
GitHub guidance: 5hrs General testing and project codebase config: 10hrs Assisting with MariaDB to Python integration: 10hrs Researching options and existing implementations: 10hrs	Researching existing bots and best practices: 10hrs Debugging, implementing chat commands and researching API: 20hrs Misc. code audits: 2hrs	Database Research/Implementation: 10hrs SQL injection sanitization: 20hrs Bot functionality with MariaDB: 10hrs Misc. code audits: 3hrs	Researching GitHub capabilities and comparing IDE implementations: 4hrs Working with API to create chat commands, debugging: 20hrs Testing existing code, researching existing options: 8hrs	Formatting meeting notes and miscellaneous research: 12hrs Researching/ Implementing MariaDB API: 6hrs Researching Python and discord.py methodology: 6hrs Debugging, testing bot, code fuzzing:

				10hrs

Milestone 2

Palmer Brown	Arturo Martinez	DJ Rice	Grace Shell	Solomon Thao
Button testing and implementation: 14hrs Discord.py research and personal testing (test bots): 12hrs MariaDB to python integration (commands and buttons): 4hrs Code audit and debugging misc.: 4hrs	Finalizing key commands: 14hrs Connecting to MariaDB: 8hrs Debugging commands and testing: 7hrs Drafting commands: 4hrs	MariaDB Schema Testing: 15hrs Bot interaction framework: 10hrs UI Display testing: 8hrs Create, Claim, Close Commands: 10hrs Misc. Code Audits: 6hrs	Discord help command: 12hrs Website editing: 4hrs Connecting to MariaDB: 4hrs Milestone 2 prep/presentation: 4hrs	Note-taking and planning: 10hrs Research paper drafting: 5hrs Button testing and implementation: 10hrs Misc. technical research: 5hrs Milestone 2 presentation and planning: 4hrs

Milestone 3

Palmer Brown	Arturo Martinez	DJ Rice	Grace Shell	Solomon Thao
Json parsing backend and robust functionality: 10hrs Functionality updates to buttons based on message context: 6hrs Persistence attempts: 4hrs ReadTheDocs work: 10hrs Github Wiki documentation: 8hrs Codebase refactoring and cleanup: 2hrs	JSON Parsing: 15hrs Code Review: 6hrs Persistence Attempts: 9hrs Final code cleanup: 3hrs	General github management: 8hrs Persistence/Dynamic Items: 30hrs Database interface changes/management: 4hrs Code Audits: 15hrs API Research: 8hrs Setup simplification/walkthrough: 2hrs	ReadTheDocs attempt: 7hrs The GitHub Wiki: 9hrs Website editing: 2hrs	Button code refinement and updates: 14hrs Note-taking and planning: 5hrs Misc. technical research: 5hrs Research paper drafting: 8hrs Bot testing: 4hrs General code updates (JSON and embeds): 3hrs

Team Reflection

Success Factors

So far, Group 2 has been commended by stakeholders for exceeding expectations, maintaining professionalism, and having in-depth analyses and comparisons of the technologies and methods at the group's disposal. Having a significant emphasis on methodological comparison allowed for two critical benefits: For one, comparison in service of finding the best fit for the team and the project shows stakeholders that their concerns are taken seriously. Secondly, this allowed Group 2 to consider strengths and weaknesses as a group, creating a better understanding of the project structure.

Further commendations for Milestone 2 include how the current iteration of the MineTicket bot exceeds the current expectations from KSU eSports. Modal implementation within the bot also proved to be a sign of proper progress and adherence to the guidelines and deliverables listed by KSU eSports.

Milestone 3's stakeholder review showed strong support from KSU eSports, the course instructor, and the project coordinator. Based on the presentation of our fully functional bot, KSU eSports representative Kylie Nowokunski described the project as "exactly what was asked for", especially giving notice to Group 2's strict adherence to all project guidelines as well as going above and beyond to implement extra features that were optional but not necessary. Group 2 focused on strict adherence to the guidelines from the project's beginning, ensuring that every week was spent focusing on critical project infrastructure development.

Team Collaboration and Communication (Milestone 1)

	General Collaboration	Meeting Arrangements/ Experiences	Collaboration Systems and Tools	Other Experiences
Palmer Brown	Team is very responsive and helpful. Worked with Solomon to create a foundation for integrating MariaDB into the code.	Meetings were productive. Screen sharing is used to great extent and usefulness.	MS Teams: Group-based communication app for meetings and general project discussion Discord: Collaboration and communication with stakeholders	I have already learned a fair bit about basic database functionality within python from this project. Sharing these findings with the team is a great bonding experience!
Arturo Martinez	Group 2 is very determined to develop a sustainable project. Group 2 fosters a positive environment to work and develop such a project.	Every meeting that we typically have includes details that must be addressed and helps keep everyone up to date. Every meeting minute is useful in some way or form.		Building the Discord bot has proven to be an enlightening experience for all group members.
DJ Rice	Team was very responsive when it was needed. Communication was clear and everyone respected the escalation path set forward at the beginning.	Finding times that worked for everyone was challenging initially, however we did end up working a schedule out that worked for everyone.		Managing the GitHub is a fairly involved process, particularly with so many different branches flying around.
Grace Shell	The team is always responsive.	Arturo and I had branched off to take on commands for the Discord. I specifically took on the help command.		Trying to understand everything about Discord commands can be a little confusing.
Solomon Thao	Group was separated into smaller teams which tackled specific problems. Personally focused on Python-MariaDB connectivity research and logging non-project details (meeting minutes, notes, etc.)	Meetings were productive and well-structured, all scheduled through MS Teams		Planning outside of the project (e.g. capstone-specific assignments) can be hectic

Team Collaboration and Communication (Milestone 2)

	General Collaboration	Meeting Arrangements/Experiences	Collaboration Systems and Tools	Other Experiences
Palmer Brown	Group 2 has continued to display favorable collaboration techniques. Communication is clear, meetings are held on time and kept to a needed duration. Group member flexibility has proven helpful.	Despite busy schedules, Group 2 has managed to arrange meetings consistently thanks to the effort of all members, especially Grace. These meetings have been productive and have assisted Group 2 in making significant progress.	MS Teams: Group-based communication app for meetings and general project discussion Discord: Collaboration and communication with stakeholders GitHub: Collaborative code-hub site in which the project is stored, edited, etc. Visual Studio Code: IDE supporting Python development with integration from GitHub	Each team member has aligned to their defined roles within the project and work effectively within those roles. The collaboration and division of tasks have allowed the group to make steady progress. Members frequently learn from each other's expertise.
Arturo Martinez	Since Milestone 1, the level of general collaboration has been consistent. Responsibility and clarity are emphasized. Group 2 is consistently working towards all predefined goals.	Despite being mostly remote, the team is flexible enough to where accomplishments and progress is clearly communicated. Though some members may be absent or late to certain meetings, proper note-taking and discussion allows for all group members to understand established guidelines and goals.		External circumstances that affect project time dedication can be easily communicated between team members. All members are dependable enough to compensate for unforeseen issues.
DJ Rice	Despite a few issues, meetings went ahead as scheduled. Certain meetings were notably productive. Members were more than willing to find the time necessary to collaborate outside of meeting times in order to meet crucial deadlines.	Meetings were efficient and informative. Group 2 is attentive during the meetings, particularly with Solomon dedicated to creating recap notes for us all to reference post-meeting. With individual breakout meetings, the team is not afraid to ask for extra help from other team members, even if they aren't present at the beginning of the meeting or initially invited to breakout meetings.		GitHub has become a more familiar platform to all members throughout this project. Pull requests serve a much more obvious purpose, and working with a collaborative team on a repo has been a valuable experience for all.
Grace Shell	Despite several issues, Group 2 took the time and collaborated closely to ensure that goals were reached during Milestone 2.	Group 2 ran into some scheduling concerns, but still found ways to make sure that each member can access the information on short-term tasks for each week.		Group 2 excels by actively supporting one another in areas where individual strengths may be lacking, thereby fostering a highly effective and cohesive team dynamic.
Solomon Thao	Group 2 is working efficiently through its proper planning and execution of properly established goals.	As mentioned, meetings are informative and have clear guidelines on what needs to be completed. In cases where further discussion is needed, notes and Teams posts are used for further communication.		Visual Studio Code has become a cornerstone of the MineTicket project's functionality, and proper usage of VSCode has become routine.

Team Collaboration and Communication (Milestone 3)

	General Collaboration	Meeting Arrangements/Experiences	Collaboration Systems and Tools	Other Experiences

Palmer Brown	Group 2 ran into a plethora of issues within Milestone 3's timeframe, but with flexible scheduling and persistent teamwork, success was acheived. Every teammate does their job diligently, and everyone's individual expertise proves to be beneficial to the group.	All meetings were clear, consistent, on time, and effective in content covered. Group 2 met multiple times in between regularly scheduled meetings to work collaboratively. All meetings were crucial in solidifying the project scope as the project continued.	MS Teams: Group-based communication app for meetings and general project discussion Discord: Collaboration and communication with stakeholders GitHub: Collaborative code-hub site in which the project is stored, edited, etc. Visual Studio Code: IDE supporting Python development with integration from GitHub	While significant work was done in collaborative meetings, each team member demonstrated a drive to accomplish the project's goals on their own time as well. Group 2's members have become much more comfortable with learning new information from other members, and has come closer as a whole.
Arturo Martinez	Group 2 pushed through the hurdles found during the completion of Milestone 3, leading to a complete product. All teammates ensured to help one another with any individual weaknesses, allowing for excellent teamwork.	All the meetings at this Milestone were always important and to the point. Group 2 never wasted meeting minutes unless there were emergencies as life still happens. All meetings ensured team members were caught up to speed with everything happening that week.		Group 2's developers truly felt the tension this Milestone as Persistence proved to be a formidable problem to solve. The team persisted through and accomplished the implementation of persistence. Group 2 has truly bonded over hardships found in the setbacks of Milestone 3.
DJ Rice	Group 2 experienced quite the hurdle with Milestone 3, being bot persistence. However, with a ton of teamwork and frequent meetings, Group 2 was able to overcome these challenges with help from every single member.	All meetings were very helpful to furthering the goal of finishing the bot, with all members making it to almost every single meeting we had. Any absences were remedied using separate one on one meetings, and the notes created ensured nothing was missed.		The group absolutely knocked it out of the park with this milestone. Tensions were high when the group discovered persistence was required to ensure the bot continued working after a reboot, but how quickly the group came together to solve this issue was immensely satisfying.
Grace Shell	Group 2 encountered various obstacles along the way, including numerous conflicts, but they remained committed to the project. Despite these challenges, we managed to overcome each barrier and delivered a successful final presentation.	Team meetings provided a platform for collaboration and problem-solving. Despite conflicting schedules and other work commitments, everyone made a concerted effort to attend and contribute to the discussions.		Group 2 faced significant challenges with Readthedocs due to compatibility issues, which required a last-minute switch to GitHub Wiki for documentation. This abrupt change demanded quick adaptation and reorganization of our content, but it ultimately provided a more reliable solution.
Solomon Thao	Group 2 ran into several major issues during the project runtime. However, due to Group 2's flexible planning structure, it was able to bypass issues through collaborative efforts centered on fixing them.	All meetings have been informative with clear weekly goals that have been consistently met. Notes and logged minutes were consistently recorded for future reference if needed by any group member.		Meetings have become spaces where the group feels comfortable with each other. The group's dynamic shifted towards a more collaborative and supportive environment as the project went on, likely due to each member growing better understandings of the group as a whole and as individuals.

Challenges

One of the major challenges of this project so far has been the presence of deliverables unrelated to the project itself, specifically when it came to planning these out for Group 2 to handle. MS Project has been immensely useful in the group's endeavors to separate tasks and assign members to them, but as for anything outside of this scope, it becomes easy to forget that there are deliverables besides the project that are needed such as weekly reports.

Milestone 2's particular challenges mainly had to do with each individual group member's time management as the semester continues. While time management was a significant roadblock for Group 2's productivity, proper discussion of scheduling and collaboration allowed for productivity to remain at acceptable levels.

Group 2 became much more organized and efficient at overcoming the non-technical issues it faced in the first two milestones. However, by the time of Milestone 3, it was unanimously agreed to be the most intensive portion of the project in terms of finding, reacting to, and implementing fixes towards technical challenges. The aforementioned persistence issue was a potentially project-killing issue if left unchecked. Thankfully, the issue was fixed after major changes to the code structure as well as Group 2's members working collaboratively to find solutions to the problem of buttons lacking persistence. The issue of persistent buttons is documented further in the "Technical Summary – Milestone 3" section.

Areas to Improve

Though no specific weaknesses were brought up by stakeholders during the Milestone 1 meeting, Group 2 has had issues in the past regarding the understanding of course material and deliverables that are not directly related to the project itself (as mentioned in the "Challenges" section). For example, an issue arose regarding the header of the project plan and status of the weekly reports. Both issues could've been easily avoided had Group 2 been more vigilant about submission guidelines.

Group 2 will need to find a way to not only have the MS Project file to reference for future and current tasks, but also assign non-project work to members. A likely solution is to discuss the non-project workweek during weekly meetings and then assign blocks of work to each member through Teams, which will allow Group 2 to reference a separate document for weekly submissions that may have missed in the process of focusing on the project itself.

Furthermore, an adherence to project management and collaboration disciplines may be beneficial to Group 2's members going forward. Though Group 2 had a flexible, comfortable, and efficient planning structure that worked for them, it did not fully adhere to processes and methods such as those found in *A Guide to the Project Management Body of Knowledge* (8), though many of these tactics applied simply through the nature of project management and execution of group work structures. In other words, Group 2's collaboration structure was effective and efficient, and though it took inspiration and guidance from project management guidelines, it did not fully adhere to them for the sake of flexibility within the group.

Appendix

Project Files

Further documentation of each file within the project can be found at the MineTicket GitHub Wiki: https://github.com/Variable-Quality/MineTicket/wiki (13)

main.py

- Holds important data such as the Discord bot Token for use in the code.
- Includes “run_setup”, which executes the setup process for the bot.

bot_manager.py

- Holds the actual Bot class
- Contains helper functions that allow for button functionality
- Contains button code, both basic functionality of buttons as well as dynamic buttons to ensure bot usage after a reset

json_parsing.py

- Contains JSON structures that allow for ticket creation, claiming, and closing through database-interactive functions.

sql.py

- Allows for connection to MariaDB database
- Contains code allowing interaction with the database by using functions as SQL commands within the MariaDB database
- Contains constructors for player objects w/ information such as ID
- Contains table-specific classes and functions used to interact with the database
- Contains several other useful SQL-related functions that allow for more streamlined database interaction

sql_interface.py

- Creates a layer of abstraction for **sql.py**, lowering risk of bad/potentially damaging inputs
- Makes helpful python objects to simplify database interactions and catch potential mistakes
- Allows for efficient staff interaction with the database while lowering security/integrity risks

- Essentially calls on sql.py code through itself

configmanager.py

- File used to import configuration file settings based on administrator editing
- Ensures a degree of customizability within the environment
- Creates a baseline configuration to be used in the event no configuration file is found

Progress Reports

Each iteration of our cohesive weekly report document is an updated version of the last. Over the 8 weeks spent on the project, there are a total of 8 documents (one document per week, both in Word and PDF format, equaling 16 files in total). Below are definitions and intricacies for the different sections of the Progress Report structure.

- **Project Status**: A color representing the weekly progress made in comparison to the week's goals.
    - **Green**: Goals were reached, with potential for progression past the weekly goal.
    - **Yellow**: Goals were just barely reached, or will require some more work within the following week to fully reach set standards.
    - **Red**: Goals were not reached, or a major setback has occurred that will require intensive attention to alleviate/fix.
- **Report Week**: Dates that signify the beginning and end of the work week.
- **Progress Summary**: A general summary of weekly goals.
- **Meeting Summary**: General summaries of meeting content. Also accounts for individual member attendance and dates of meetings throughout the week.
- **Member Activities**: A loose tracking table of what each individual member has worked on throughout the week, including short descriptions and approximate hours spent on different tasks.
- **Total Team Hours**: Sum of all individual group member hours spent on project work for the week.

Example of Weekly Report

Project Status: Green!

Report Week: 03/24 to 03/31

Progress summary

Milestone 2 is now behind us. So, we're putting the finishing touches on our bot and making sure its up to spec with the sponsor requirements. Primarily, we're working on the automatic systems such as parsing incoming json text from a discord channel, which will be made into a ticket without any user interaction.

Meeting summary

03/25: Sponsor meeting, informal

Members in attendance: Arturo, DJ

Clarified expectations on ticket archiving, information that will be given to the bot for tickets, and the API requirement set forward.

03/25: Team Meeting, informal

Members in attendance: Palmer, Solomon, Grace, Arturo, DJ

Discussed best way to proceed with implementing necessary features, re-divided the work to ensure the important things have ample manpower.

03/27: Team Meeting, Informal

Members in attendance: Palmer, Grace, Solomon, DJ, Arturo

Checked in on existing progress for code, performed some maintenance on the github to ensure all branches were properly updated.

Member activities

Solomon Thao	1. Code implementation (embeds, JSON cleaning)	3hrs
	2. Work sessions w/ DJ (button functionality updates)	5hrs
	3. General research (persistence) and documentation/notetaking	2hrs
Palmer Brown	1. First implementation of persistence	4hrs
	2. Backend updates for readthedocs w/ grace	4hrs
	3. Personal testing/updates for readthedocs	5hrs
	4. Updates to linkedin for C Day	1hr
Arturo Martinez	1. Persistence Session #1 w/ Palmer (First implementation)	4hrs
	2. Persistence Session #2 w/ DJ (Dynamic implementation)	2hrs
	3. Persistence Session #3 (personal implementation via existing py bots)	3hrs
Grace Shell	1. Weekly meeting/prep for milestone 3	2hrs
	2. Read the docs (research/trial & error stuff)	4hrs
	3. review website for Milestone 3	1hr
DJ Rice	1. Github management	2hrs
	2. Researching ReadTheDocs compile problems	2hrs
	3. Researching discord persistence/dynamic buttons	6hrs
	4. Reworking code to fit persistence requirements	20hrs
	5. Debugging code/edge cases	3hrs

Total Team Hours: 73hrs

Bibliography

1. *Deploy MariaDB Community Server 10.6*. MariaDB. (n.d.). https://mariadb.com/docs/server/deploy/topologies/single-node/community-server-10-6/
2. *MariaDB Connector/Python*. MariaDB. (n.d.-b). https://mariadb.com/docs/server/connect/programming-languages/python/
3. *Introduction*. discord.py. (n.d.). https://discordpy.readthedocs.io/en/stable/intro.html
4. *A primer to gateway intents*. discord.py. (n.d.). https://discordpy.readthedocs.io/en/latest/intents.html
5. *Cogs*. discord.py. (n.d.). https://discordpy.readthedocs.io/en/stable/ext/commands/cogs.html
6. R, T. K. (2020, January 18). *What's this weird arrow notation in python?* Medium. https://medium.com/@thomas_k_r/whats-this-weird-arrow-notation-in-python-53d9e293113
7. GeeksforGeeks. (2023, March 24). **args and **kwargs in python*. GeeksforGeeks. https://www.geeksforgeeks.org/args-kwargs-python/
8. Project Management Institute. (2021). *A Guide to the Project Management Body of Knowledge (PMBOK® Guide) – Seventh Edition and The Standard for Project Management (ENGLISH): Vol. Seventh edition*. Project Management Institute.
9. *Commands*. discord.py. (n.d.). https://discordpy.readthedocs.io/en/stable/ext/commands/commands.html
10. *How to sync slash command globally discord.py*. Stack Overflow. (1968, September 1). https://stackoverflow.com/questions/74413367/how-to-sync-slash-command-globally-discord-py
11. *Buttons*. Pycord Guide. (n.d.). https://guide.pycord.dev/interactions/ui-components/buttons
12. *The cursor class*. MariaDB GitHub. (n.d.). https://mariadb-corporation.github.io/mariadb-connector-python/cursor.html
13. Brown, P., & Shell, G. (2024, April 19). *MineTicket Wiki.* GitHub. https://github.com/Variable-Quality/MineTicket/wiki