Discord Focus Bot Final Report

The Discordians

Authors

Shreya Aravindan
saravindan@vt.edu

Geethika Abhilash
geethika@vt.edu

Sheril George
sherilhannah@vt.edu

Justin Turkiewicz
justint417@vt.edu

John Byman
johnbyman@vt.edu

Arav Tripathi
aravt@vt.edu

ABSTRACT

The Discord focus bot is a programmable bot configured for the Discord social platform to assist in various tasks for Discord servers [1]. The bot offers features such as managing tasks from a dedicated task board on a discord channel, setting and receiving reminders about events, monitoring notifications of activity from a git repository and using a pomodoro timer for studying. The focus bot intends to be an all-in-one bot that provides everything a discord server needs for automation purposes. Managing, setting up and learning the commands issued to one bot to do different types of tasks creates a simpler and streamlined process for Discord server admins to maintain and use.

INTRODUCTION

Problem

Software engineering teams often face challenged in effectively managing deadlines and tracking code changes withing their projects. Discord adds to the communication burden for software engineering teams. Manual tracking of deadlines and code changes leads to inefficiency. There is a need for automation to streamline communication and enhance productivity.

Proposed Solution

Our proposed solution is a discord bot. Discord bots are software applications that perform various automated tasks within Discord servers. Our bot is planned to streamline communication and enhance productivity amongst teams of software engineers. The bot facilitates the creation of a task board within discord and can be used to set reminders in the form of discord messages. The bot automates reminders for deadlines, reducing the risk of missed milestones. The bot also has Git integration which sends real-time notifications for code changes, which enhances collaboration. Lastly our bot should support the pomodoro technique.

RELATED WORK

Discord bots in general are not a new technology, so there were many sources we referenced for designing our bot. The first source we looked at was an article called “The best calendar bot for discord” for the Sesh discord bot [4]. The second one was an article for a multipurpose discord bot that sends alerts for various events. The bot was called “Lion Bot” [5]. The last source is a github repo for a bot that is configured to send notification for git actions [6]. We chose these articles because our bot is planned to employ a variety of the features of the bots described by these sources.

IMPLEMENTATION

High Level Design

For the high-level design, we decided with an event-based architecture for the design of the discord focus bot. The event-based approach is ideal for the bot as users input commands that get sent to the bot and the bot must process the commands to produce the desired output. For example, creating a task on a task board would require the bot to know which task board the task should be added to base on the user’s input in the command to create a task on a specific task board. Thus, an event-based architecture is suitable as the discord focus bot relies heavily on user actions and input to perform its tasks correctly. Some events the bot must monitor as well. When users set a reminder for an event or create a pomodoro timer through the bot, the bot must be aware of the time when it should response with a reminder or that the pomodoro timer has ended. There are internal mechanisms within the bot’s code that check when these events occur.

Implementation Process

We went with a behavior-driven development process for creating the various features of the discord focus bot. We felt that usability and the user experience are crucial for using a discord focus bot. Convenience and easy to understand processes for users to follow to use the bot are paramount to gain user adoption and satisfaction of the discord focus bot. First, we designed mock user interfaces in the form of storyboards to decide on the look and feel of the discord bot. We paid close attention to the semantics and tone of the bot’s response to user commands. The bot’s language was designed in a way to feel satisfying to interact with and contains positive language when user issued commands fail or do not do

what the user thought it would. These aspects are crucial in creating a rich and exciting experience interacting with the bot such that users are encouraged to come back and use the bot again.

After deciding on the mock user interfaces for the various features of the discord bot, we went ahead with the implementation portion of the project. This phase encompassed writing test cases that the bot would be designed to pass. We write the tests before writing the code so we can view and agree on how the bot will function from a user perspective. Finally, we write the code to pass the tests.

Testing Approach

When designing the tests for the code, we decided that there would be unit tests generated for each individual command that can be made to the bot. This confirms correct and expected functionality of the different pieces and functions of the codebase. Some commands require context or previous actions be completed before successfully executing them, such as the /startPomodoro command being a prerequisite to the /pausePomodoro command in order for there to be an active pomodoro timer to pause. In addition to unit tests, there were integration tests that verify correctness of the full user flow of the various features that the bot supports. For example, there were tests that ensure that a user in succession can create a task board, update the task board's settings like its font size and title, create a task on the taskboard, update the task (e.g. who it is assigned to), delete the task and then finally delete the task board. These comprehensive integration tests give confidence that the discord bot can perform all functions of a given feature such at these functions work in context of prior events. Finally, we included system tests so we can confirm that the discord bot can perform actions simultaneously across various feature sets. We developed tests that ensure different feature sets can work at the same time when using the bot. Examples of such tests include having the discord bot track reminders set by a user and then subsequently start a pomodoro time by another user. These tests confirm that the bot can perform a wide variety of tasks at the same time.

DEPLOYMENT AND MAINTENANCE

Deployment

The Discord focus bot project will use test automation to solidify the quality of the bot's functionality before the bot is deployed or a new version of the bot is released. Test automation will guarantee that the bot can perform the functions based on the series of events within the provided tests. Test automation frees up time and energy that otherwise would have gone to testing the bot manually. Instead, developers could focus on fixing bugs with the bot or adding more features to the bot.

To further minimize bugs and facilitate testing in scenarios, continuous integration will be used along with test automation. Continuous integration focuses on building and testing all code changes submitted to the main branch. This will help in identifying conflicts and bugs early in the development cycle, which simplifies debugging and improves code quality.

When the bot is deployed or a new version of the bot is released, a Canary Deployment system will be used. The canary deployment method ensures that new versions of the discord bot are introduced incrementally to a subset of users before a full roll out. This approach will reduce the risk associated with deploying new features by allowing us to monitor the bot's performance functionality with real users under real conditions. If any issues are detected we can quickly roll back the changes without affecting all users, thereby minimizing issues.

Maintenance

The Discord Focus bot will be maintained through continuous refactorization of the code, code reviews, and using debugging tools to repair problems and bugs within the project's codebase.

Refactoring the code for the bot will improve the readability of the code. Refactoring also will likely reduce the size of the codebase as simpler and more scalable ways to design the code are introduced into the codebase. Whenever functionality is added, bugs are fixed or code is refactored for the bot, code reviews will be performed.

Code reviews will ensure that multiple people view the code changes which will likely improve the quality and certainty of the code changes. Code reviews also give developers a view and become familiar with other portions of the codebase allowing them to work on other portions of the Discord Focus bot project.

For debugging issues found in the project, we will use automated debugging tools such as VSCode's built-in GUI debugger [3]. This debugger eases the process of debugging code within the application as it conveniently displays data based on the current line the code is break pointed at. Examples of data displayed at the break point would be variables that have been declared at that point in time including the values that these variables store, the call stack and a debug console to display debugging logs.

CONCLUSION

Limitations

The Discord Focus Bot is not without certain limitations. One such limitation is rate limits. A bot can only perform actions at a certain rate and if too many commands are sent too quickly, the bot may slow down and stop responding. Another limitation is that the bot is internet connection dependent. If the user is offline, the bot will not work as it requires a stable internet connection. The bot also has limited multitasking capabilities. The bot may struggle to do any tasks simultaneously, which could lead to a slower response time. Lastly, users may experience a learning curve as they need to learn specific commands to effectively communicate with the bot to utilize its features to its full potential.

Future Work

In the future, we want to expand the usability of the bot by introducing additional features that enhance the user experience. One such feature is the Focus mode, designed to help users concentrate better on their tasks. Using Focus Mode, users will be able to mute notifications and hide distracting channels, creating a better working environment. Another feature we hope to add is the bot's seamless integration with other productivity tools, like Google Calendar. Using the bot, the users will be able to schedule focus sessions, create events, and receive reminders directly within their calendar. Another feature the bot will offer is Progress Tracking. Using the bot, users can access detailed statistics on their focus sessions and completed tasks. The user can benefit from this feature as it will let them understand their work habits and identify areas of improvement. Lastly, we wish to enable the bot with abilities to provide the user with Progress Visualization. Using the bot, users can visualize their productivity trends over time through interactive charts and graphs of completed tasks. Overall, these features aim to give users a more enhanced and personalized experience, with the goal of helping them stay focused, organized, and productive.

REFERENCES

[1] Discord, https://discord.com/.

[2] T. Mandel, “Chapter 5 The Golden Rules of User Interface Design,” in The Elements of User Interface Design, John Wiley & Sons

[3] Microsoft, “Debugging in Visual studio code,” RSS, https://code.visualstudio.com/docs/editor/debugging (accessed May 2, 2024).

[4] “The best calendar bot for discord,” The best calendar and event bot for Discord, https://sesh.fyi/.

[5] “The best discord bot,” LionBot, https://lionbot.org/ (accessed May 2, 2024).

[6] Falconerd, “Falconerd/discord-bot-github: Github repo updates displayed in discord.,” GitHub, https://github.com/Falconerd/discord-bot-github (accessed May 2, 2024).