AlarmBot

Set-up and initial plan

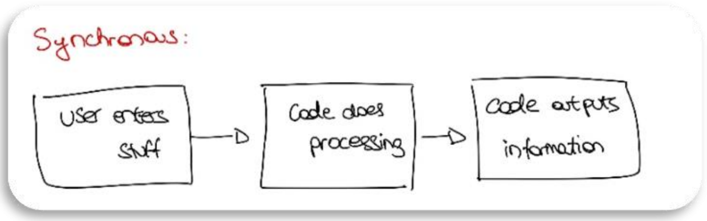
I have not coded a Discord bot prior to this project, so I researched how to set up a Discord bot in the first place with Python. As I found out, I needed to do two things: setup a link between the Discord server and understand “discord.py”: a library that is necessary to code in Python for adding code to the bot.

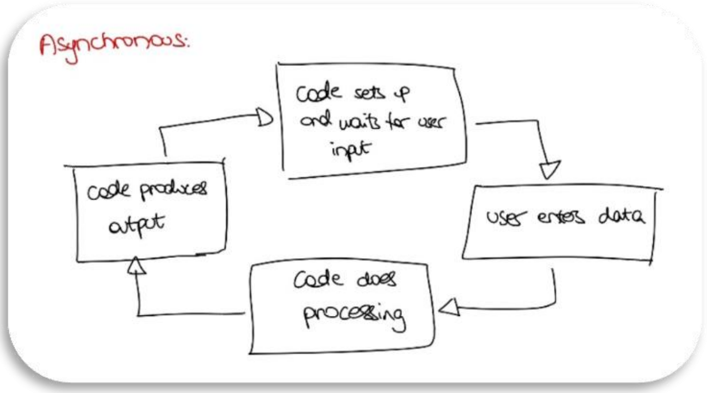
Linking the server and the bot

Using the “.env” file, I stored the Discord bot’s unique token into it, so that the code does not expose the token when exported by itself. Access to this token allows anyone to manipulate the bot, and by extension the server it exists in, at will.

Understanding discord.py

Python is a programming language that aims to “simplify” coding by doing a bunch of instructions when given much fewer somewhat readable/understandable commands. Becoming very popular, it spawned “libraries”- collections of code that people produce so that others can build on these. One of these is discord.py which allows python code to directly execute instructions relevant to Discord. It relies on a concept called asynchronous programming.





Something unique to asynchronous programming is the ability to have multiples of these cycles running at one time, making it quite powerful, as tasks don’t need to wait for each other to finish. Since we’ll be needing to constantly have the bot be running to keep track of everything, it needs to run in this asynchronous manner.

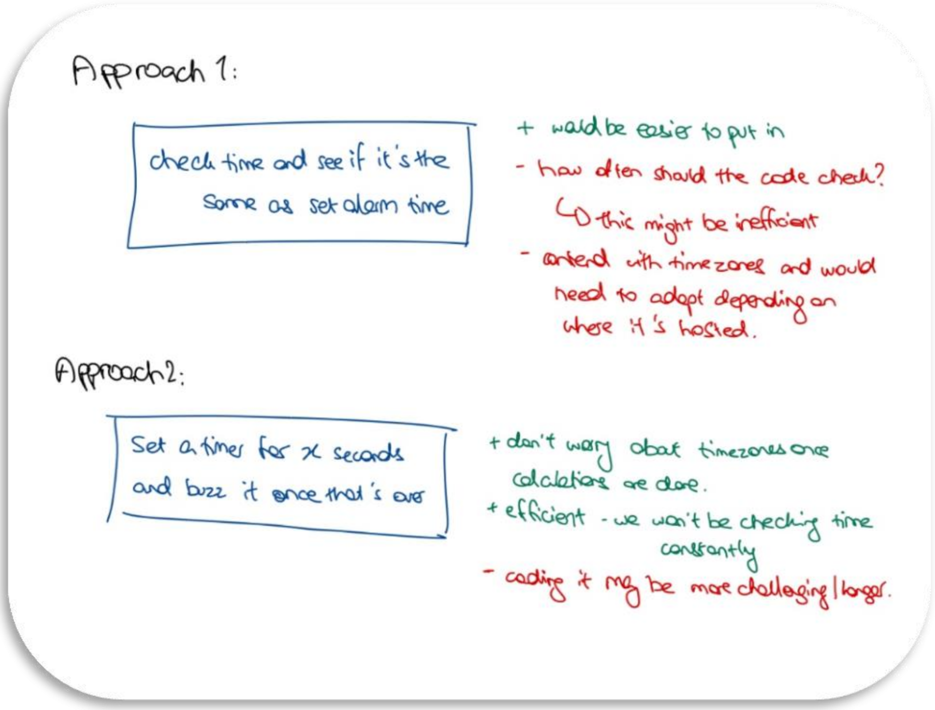
Implementing functionality

Progress on the tasks given was incremental.

The process began with first implementing “/alarm”. From reading ahead I knew to process the word “set”, so I added a “command” parameter (which would be the “set”) and the time, both of which default to

“None” if they aren’t provided. So, if someone sent “/alarm” they would hit the criteria and the bot would ping everyone.

The next step was to start incorporating the alarms themselves. For this, I needed to be able to convert from a user given HH:MM time to a “datetime” format, a form of time that Python can understand and process. To do so, it needs the date but also the seconds as well. For this purpose, I used regular expressions, strings of characters that can match only the HH:MM, and then attached “00” seconds as well as the current UTC date to convert. Now that we have the target alarm time, there were two ways to “do” the alarm itself:


Approach 1:
check time and see if it's the same as set alarm time
+ would be easier to put in
- how often should the code check?
↳ this might be inefficient
- contend with timezones and would need to adapt depending on where it's hosted.
Approach 2:
Set a timer for x seconds and buzz it once that's over
+ don't worry about timezones once calculations are done.
+ efficient - we won't be checking time constantly
- coding it may be more challenging/longer.

The obvious choice was to pursue approach 2, justified by its smaller drawbacks. This was done with UTC timings in mind so that no matter where the bot was located, it would still use the same timezone for calculations, which could be offset by the user later. The command “/time” made a lot of sense here for that offset, and to keep consistency with the overall task it took a similar format as well. An advantage of using this over timezones is that users could specify exactly how much of a time difference they want, as well as maintaining privacy in case they don’t wish to specify their timezone and by extension their rough geographical location.

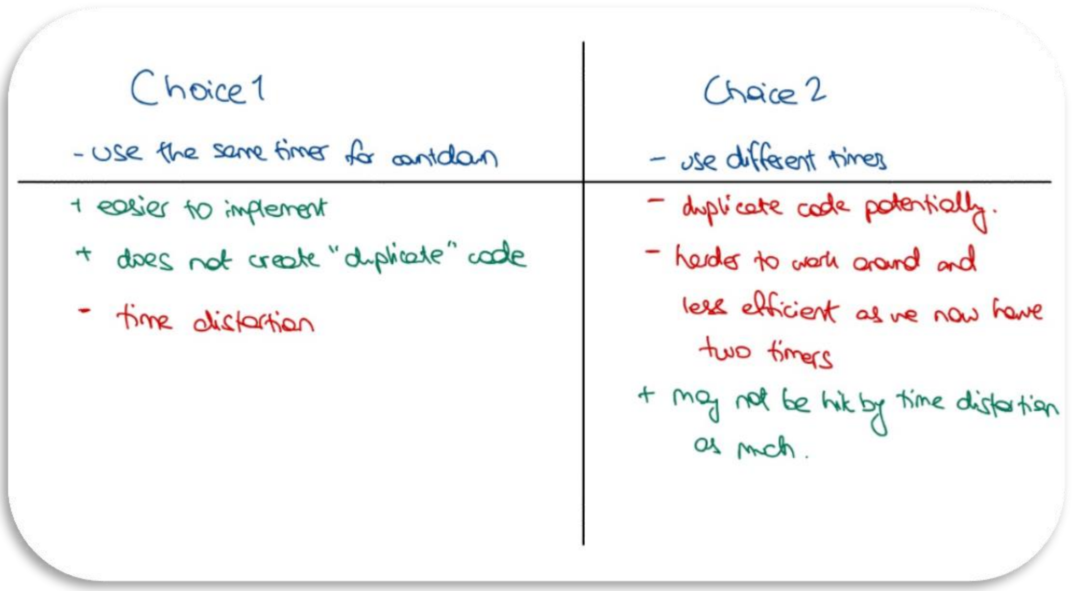
Since the requirements specify to type “/alarm set 00:00” to set an alarm without the choice of a day meant that I could safely assume that alarms set “backwards in time” could be triggered the next day, i.e., setting an alarm for 9:00, given that the time is 10:00 meant setting an alarm for 23 hours.

A similar process had to be applied for the current time offset applied by the user: if the time given is in the “past”, then this meant that the user required a negative time offset. An important detail in calculating this is that I checked the difference between the time the user gave and UTC time was greater than 12 hours. In doing so, we potentially exclude differences of more than 12 hours either way from UTC, which is an oversight and something to be solved in a future iteration.

A small addition to the code came in the form of the toggle feature which takes no values, just the command by itself. It made sense to declare this toggle as a “global” variable, something that all the code blocks could see, so that the alarm would only trigger if this was “on”.

Further development took place to ensure that all functionality worked as intended, understanding that core functionality like /alarm needed to be revamped to accommodate for the toggle and how /alarm interacts with "/alarm set HH:MM" i.e., we want to trigger the alarm if it is toggled on and make sure that entering /alarm again doesn't trigger it a second time.

One of the last features to be added was the countdown, which I reasoned would be better if it was separate from the actual alarm timer itself. The code at this point calculated the difference between the alarm and the current time to generate the required seconds to "sleep" (do nothing) for. If I used the same timer for the countdown that the alarm used, then theoretically, it may "miss" the time it was set for since in a loop, it would subtract 1 second from the given numbers of seconds, then "sleep" for one second. The time required to perform -1 would slowly stack up over longer periods of time leading to minutes of inaccuracy. As such, it made sense to trade this for an option where I would make the alarm function as accurate as possible, while the new timer for the countdown could be inaccurate but would likely be a good indicator of when the alarm goes off.



Choice 1	Choice 2
- use the same timer for countdown	- use different timers
+ easier to implement	- duplicate code potentially.
+ does not create "duplicate" code	- harder to work around and less efficient as we now have two timers
- time distortion	+ may not be hit by time distortion as much.

Once this was wrapped up, an important aspect was testing and covering user input through validation, which led to smaller changes to make sure that all combinations of features were being tested accurately.

While the only extra functional command added was "/time" to account for the user's timezone, I believe in making existing code function as well as it possibly can before adding new features. I've chosen to add details to the /countdown feature, where it neatly describes the hours, minutes and seconds left in the alarm. Alternatively, it could've easily been made to return "HH:MM:SS" time, but it would've spoiled the user experience, seeing something like "07:23:44 left till alarm goes off". I've used this approach across the board to create a bot that answers the brief as closely and well as I believe it can.

Thank you for reading 😊 have a great day Henry,

Vedant