Overlord

Overlord: A Fully modular and selfhosted discord server moderation and management platform Using Discord.js.

This takes the form of a Bot user, as well as a shared Backend, and Website.

The bot will be coded using Node.js, as will the backend – the Database will be a persistent ENMAP instance (Using SQLite). The Website will also be written Node.js, supplemented with Vue. This entire project will be Hosted on my server(s) on a custom Domain, although I may decide to place it all into a docker image for ease of use and selfhosting. The project is all within a Private GitHub repository that may become public when the project completes. It will use various other dependencies for certain functionalities.

For those unaware, Discord is a massively popular communication app, that allows for ‘servers’ that contain both text and voice channels. Users can join these servers and are managed by a rather complex permissions system as to fine tune what users can do what where, oftentimes governed by ‘tags’ called roles. Due to the large numbers of users, manual moderation quickly became inefficient. In response, people began developing pseudo-unofficial API’s to allow for automated users (“bots”) to access discord and to do these tasks instead. Overlord is one such project, using the API Discord.js.

Analysis:

As a server administrator and a basic user on many discord servers both large and small, I have determined there are a few core features that my bot must perform (and do well at!) in order to succeed.

These key features are:

* Auto moderation. This is the primary reason most discord bots exist. There are many categories of content that the bot will have to detect and then programmatically deal with.
* Configurability: almost all aspects of the bot’s behaviour should be able to be controlled on a per-server-per-channel, as well as a per-role-per-user basis. This configuration should be as easy as possible for less experienced server admins, but also powerful enough to enable advanced customisation for those who want it.
* Uptime: what’s the point in having a bot if it goes offline constantly? This is one of the easier problems to solve utilising the Node Module PM2
* Modular command processing + response management
* Automated Management/moderation of User activities with user and server specific configuration to dictate bot behaviour
* Web interface / management UI for Server Admins/moderators.

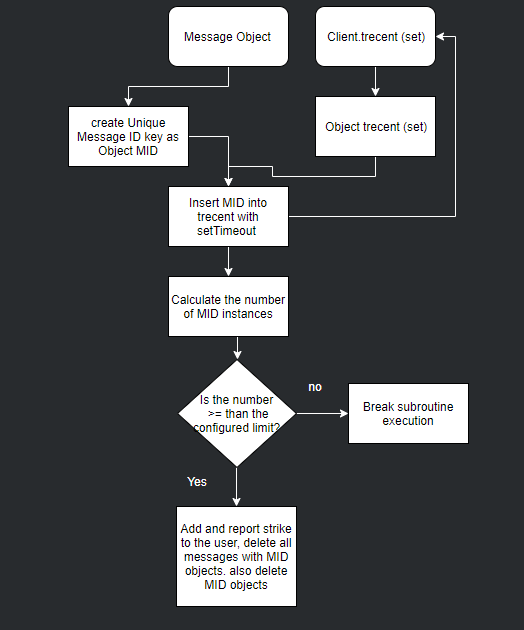
Module/expanded feature list:

* AutoMod: Detects, removes offending content, and penalises the author of the content (eg, spammed messages.)
* Ability for the bot to store and return programmed responses, eg the command $Image1 would cause the bot to send whatever file it has associated with the name ‘Image1’ on a per-server basis
* Message editing/deletion monitoring and reporting – as discord has the ability for uses to remove/edit content ‘silently’, this feature is important for proper effective moderation.
* Ability for users to report bugs through the bot, and the ability for the bot to autodetect and report errors.
* Resilience features – to protect against crashes due to unexpected errors, and to allow the bot to auto recover from crashes automatically

Other Miscellaneous features:

(these features are not vital to the main role of the bot, but would be useful to have)

Development Log: Auto Moderation Algorithm Planning:



Developmental log: Database Schema

This bot has a \*lot\* of configuration options, and to help me (and you) get our heads around it, I utilised a bit of pseudocode to provide a object tree of what data would be stored in the Client, and where.

Validating User Permissions.

In this bot, users can have permission levels ranging from 1-4, where 1 is any (non-banned/penalised) user, 2 is any user with a Moderator role (specified in guild configs), 3 is any user with a Administrator Role (again specified in configs) and 4 is Bot owner, which in this instance will be me.