Chris Lodge

CS495

2 November 2019

Senior Project Paper

For my senior project I created a Discord bot in NodeJS. This bot's role is to provide server management and convenience functionalities to a Discord server. Discord is a commonly used chat client that is a mixture of direct messaging and chat servers. Users can join as many different servers as they would like where they can communicate in text/voice channels with other server members. Another large aspect of Discord are the multitude of bots that have been created to do things ranging from responding to messages as famous people to tracking statistics in video games to providing in chat games. One of the large questions that this poses is what are the best technologies to use to create one.

When first looking into what technologies to base my project around, I began thinking about what similar projects I have done in the past. This led me to think about using either PhP, Node.js, or flask/bottle in Python. Due to my inexperience in Python, I decided to narrow it down further to PhP or Node.js. Since my familiarity with both of those languages is about the same, I started to look at what support was already in place for working with Discord in either language. This led to me finding discord.js, a Node.js module that wraps all of the required api calls in an object oriented wrapper. On the other hand, I also did not want to have to deal with all of the apache set up that using PhP would require. Ultimately I decided on Node.js because I felt like I could structure it a lot better with the discord.js wrapper than if I had to build the Discord api interface from scratch. This choice would later prove to be invaluable as Node.js also has packages for turning YouTube videos into audio streams. From there I went on to think about what services I wanted my bot to connect to.

One prevalent feature of most Discord bots is the ability to access numerous web apis through them. For my bot I decided that it would be best to have a mixture of useful and humorous ones as at the end of the day Discord servers are not exactly serious places. I figured that a weather api would be a good place to start. This proved to be a challenge, as I had never accessed a web api programmatically before. However, once I figured out how to attach credentials to my api calls, it became much easier. From there I continued to add access to a bunch of other simple apis such as trivia, fact, and joke ones. These were relatively simple to add as only some of them required parsing user input beforehand, even though I did have to format all of their responses from json to something presentable. However, one that I failed to find anything for was a dictionary based one. I kept getting authorization failed errors on most of the big ones that I tried. Little did I know, some apis actually block requests from javascript because they do not want api keys to be based client-side. While a lot of apis took into account that some javascript is run outside of browsers, unfortunately this did not include any of the dictionary ones that I could find. Instead, I figured that I would have to simulate that functionality on my own.

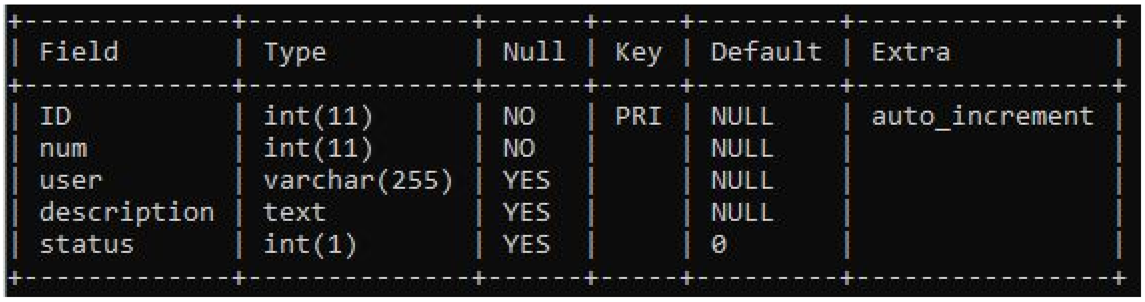
Before I go through and explain my solution, I just want to mention that I know that there are much easier ways. I could have just grabbed one of many Node.js libraries that provide static references for the vast majority of words that users could come up with. However, I wanted to provide a much more creative solution instead. I started by looking through Wikipedia and seeing if their web pages had any “too long did not read” kind of summaries to all of their articles. I ended up finding that in the raw Html on most of their articles there are a lot of unused sections that are normally hidden. One of these such sessions is called the ‘short description’ which on most articles provides a small snippet that summarizes it. This fact combined with the fact that Wikipedia has articles on the vast majority of things proved to be the solution to my dictionary problem. I ended up just parsing the user input into whatever the Wikipedia url for that thing would be and then scraping the web page and taking the short description section out and returning that to the user. This issue of not being able to find straightforward apis continued as I tried to add built in programming documentation functionality.

Trying to add programming documentation functionality was one of the hardest parts of this project. I started out with wanting to add this functionality for Java, PhP, and Html, as I felt that those were some of the most well documented languages. While I was unable to find an api to provide this kind of information I figured that it would be easy enough to just scrape from a webpage, however I was wrong. I started with HTML as I figured it would be a good starting point due to the fact that it would essentially just be a tag lookup. I found that even if I could modify the search url to add in search terms dynamically, the method that I was using to grab the web pages could not correctly follow the redirect. In order to get around this, I ended up just manually constructing the url for the relevant tag. If the url was not valid, I would inform the user that it was not a valid tag. If the url was valid, I would scrape the summary of the tag from the page and present it to the user. This was possible because they used consistent labels for each section of information, as one would expect from a Html reference site. However, this was not the case with PhP and Java.

While I had the same redirect error with PhP and Java that I did with Html, I was unable to find a work around similar to the one I used for Html. For PhP and Java, the main issue was that there are a multitude of sub categories for functions/classes to be under. This would not be an issue if I was able to use their built in search function, however since I could not follow their redirects I had no luck with this. I figured that I would play around with the idea that I could find something to make that possible anyways. However, the PhP documentation pages do not have a standard format, making it impossible to grab the relevant information without just grabbing the entire webpage. Even if I did just grab the entire webpage, there was no way that I was going to be able to carve away all of the hidden text/menu/UI elements that would also be included in it. I didn't even get as far as this thought experiment when it came to Java.

Java was the last language that I expected to have trouble with as its documentation is one of the main reasons that I like it so much. However, I very quickly realized that I had never had to use the search functionality that is baked within the documentation, I would always just use Google to find the relevant page. The only kind of search functionality that I could find was the overall Oracle documentation search. It seemed like this wouldn’t be a problem as I was able to restrict the category to Java, however it searches all Java resources in their entirety. What this means is that it searches every version of Java, every user guide, and all sorts of other articles and brings back anything containing the search query in either the title or in the entirety of the page. This makes it impossible to just rely on the first search result as it may be completely unrelated to what the user was trying to search for. This issue alongside all of the others that stopped me from being able to do the same for PhP, caused me to give up on adding it to the bot. With that wrapping up the api portion of my bot, I moved onto other features.

One thing that I wanted to make sure to highlight with my bot was the ability to utilize persistent storage. I decided to use MySQL for this due to my previous experience in it and the fact that I knew that I could access it from within Node.js with relative ease. A relatively practical feature that I chose to highlight this with was a todo list. The table structure is as follows.



```
+-------------+--------------+------+-----+---------+----------------+
| Field       | Type         | Null | Key | Default | Extra          |
+-------------+--------------+------+-----+---------+----------------+
| ID          | int(11)      | NO   | PRI | NULL    | auto_increment |
| num         | int(11)      | NO   |     | NULL    |                |
| user        | varchar(255) | YES  |     | NULL    |                |
| description | text         | YES  |     | NULL    |                |
| status      | int(1)       | YES  |     | 0       |                |
+-------------+--------------+------+-----+---------+----------------+
```

One difficulty that this posed was the fact that Discord does not require usernames to be unique. However, behind the scenes they do still have a unique identifier that I can access from their message. Everything else in the table is relatively straight forward, the only issue that I ran into was making it so that the todos are easily accessible so that users can complete/delete them, but all that took was me adding in another ID field that counted on a per user basis (the ID assigning logic was kept on the Node.js side for simplicity's sake). After I knocked that feature out, I started working on the server administration commands.

One of the big upsides to Discord is just how much customization they support when it comes to controlling servers. The main way in which this is accomplished is roles. Each role has a certain set of permissions, controlling what administrator stuff they can do, what channels they can view, and what custom commands they can use. However, one downside to this is that there is no system in place for automating roles due to the fact that administrator permissions are required to assign a role to someone. While this may seem like a good idea from a security perspective, many servers use roles in order to organize what people want to see what content.

This is where bots come into play, since administrators can use them to allow a certain subset of roles to be assigned to users via chat commands. This seemed relatively straight forward until I realized how difficult it was to find users/roles by name. I ended up trying to manually loop through the lists of users and roles and select from there, however that did not deal with the issue where multiple users can have the same name and seemed like an overall bad solution. Luckily though, Discord has a built in functionality where roles and users can be referenced with # and @ symbols respectively. This brings up a small in line interface where users can select exactly what user/role they want to mention. These get attached to the message as two seperate arrays that I ended up accessing to get the user/role object from. This made it incredibly easily as all I had to do was send a few function calls to those objects. With that done, the final thing to do was add audio playback.

One of the most popular reasons for wanting a Discord bot is to play YouTube videos through voice chat for comedic effect or as background noise. The biggest issue that this poses is actually getting the content from YouTube in the first place. Once it is cached locally, it is not very hard to create an audio stream with it using FFmpeg, another Nodejs package that focuses on audio/media related functionality. I ended up settling on using Opus Script in order to grab the videos from YouTube, however since Opus ends up simply using Python in the background I also had to set up Python as well. The only catch with doing it this way is that YouTube has started IP banning any bots that send a large amount of requests, however for the purpose of this project there is no way that I am going to end up reaching that limit so there is not need to try to implement a work around. Once I figured out how to get them cached, the rest of it was relatively straightforward. With that, the project was done.

Throughout the entire project I learned just how difficult it is to get multiple technologies to work together. The vast majority of my time was spent looking through outdated documentation and wondering why example code would not work. This was exacerbated by all of the different protocols and authentication methods that each of the apis used. It did not exactly help that this was my first time actually accessing api endpoints without using any kind of wrapper code. This frustration ended up forcing me to pass on using certain apis and using suboptimal ones, but on the flip side this made me realize just how vast the collection of apis out there is. It was even more surprising just how many of them are open to public use without any kind of authentication. However, on the flip side it made me realize that ultimately a lot of apis will come and go, especially if they are just small ones made by people as jokes with no real incentive to keep them up other than a hobbiest mentality. Another upside was that ultimately this was exactly what I wanted to learn from this project, the process of getting a bunch of apis to work together. My main motivation for wanting to learn this was always hearing about just how much easier apis can make projects and I felt like I was unable to benefit them from due to my lack of experience with them. It will definitely make me much more open to using apis in the future.

Ultimately, I had very mixed feelings about the difficulty of this project. On one hand, I did not end up with very much code, but on the other hand, I spent a lot of time on things outside of coding. Due to my lack of experience with apis, I had assumed that they would be complicated to access from a coding standpoint, but that was not necessarily the case. It turned out to be much more work before even getting to that point, reading through documentation, getting authentication methods right, and correct formatting. If I had to do this project over again, I would have chosen a project in which I had a lot more control over every aspect of the project so that I could introduce more custom design ideas. I also feel like this would have caused it to be a lot bigger from a lines of code perspective because it would be more of a ground up kind of project instead of building on top of an existing package.Ultimately though, I was still happy with how my project turned out and I see myself personally using it and adding to it in the future.