# **Senior Project Paper**
# Discord Chat Bot: Asynchronous PHP

BENJAMIN HARRIS
*Northern Michigan University*
April 25, 2017

## Introduction

I chose to build a chat bot for my senior project. It started as a side project that I happened upon while chatting with friends in a Discord chat. I started spending a lot of time on this project as I added more and more features to the bot.

My original plan for my senior project was to make a social beer rating site as an optimized mobile website (leveraging the latest browser tech to make it feel native). However, the more time I spent working on the bot, the more I dreaded starting on this site. I ended up deciding that I should commit to the project that I was more passionate about. After submitting a new project proposal, I continued work on my project.

My bot now has over 2000 lines of code and is able to fulfill many generic purposes that one would look for in a large chat group. The bot is now being used in over a dozen public Discord servers.

## What I Learned

The biggest thing that I've learned is that being excited about the project you're doing makes everything a whole lot easier and more fun.

In terms of tangible skills, I have learned a great deal about writing asynchronous code, as well as modern, style-compliant PHP, following the PHP Standards Recommendations by the PHP Framework Interop Group (FIG).

I also learned a lot about the Standard PHP Library (SPL) and the cool features included in it. I especially liked the interfaces defined by the SPL that provide easy solutions to common problems. I like the various Iterator and Array Access interfaces. Combining these

two interfaces, I created my `PersistentArray` class. Since the only data that the bot would need to store are simple key/value pairs, I decided to access them as associative arrays and writing a serialized version of that array to file. It was a fun learning experience to implement something from the SPL.

## What I would do differently

The main thing that I would have done differently would have been to choose the framework to use based on something other than familiarity with the language. While I made the project quite hard for myself, I feel that some of the things that I built for my bot would have been much easier to complete in a different language or with a framework that has a larger support community (see the DiscordPHP section under Technologies Used).

Something that I would like to look into in the future is Redis. I've read a bit about them and they seem to be better-suited to my use case of key-value pairs than a traditional database.

## Technologies Used

- Discord API
- DiscordPHP API Wrapper
- ReactPHP Promises
- Guzzle PHP HTTP client
- PHP 7.1
- Ubuntu Server 16.10
- Composer Dependency Manager (PSR-4 class autoloading)

I chose to use PHP for my bot when I originally looked at creating a bot for Discord. There are API wrappers for most major programming languages. I felt the most comfortable in PHP because I've had the most experience with it.

DiscordPHP is the wrapper in PHP for the Discord HTTP/WebSocket API. It boasts nearly full coverage of the API, even including voice support for streaming audio to a voice channel within Discord. The downside to using the DiscordPHP framework is the poor documentation and relatively weak developer community. There is a group on Discord itself dedicated to DiscordPHP, but it is rarely active and I already find myself knowing

more than most of the other users there. Additionally, the framework authors are seldom seen and have not resolved outstanding pull requests or made any new contributions on GitHub in over two months.

Since PHP is inherently synchronous and single-threaded, we need to account for the asynchronous nature of responding to commands and messages in a chat. Promises are a way to deal with operations that take differing amounts of time to complete. Promises represent the eventual value returned from the completion of an operation, and can be unfulfilled, fulfilled, or failed. The DiscordPHP framework uses the ReactPHP implementation of Promises to deal with asynchronous events.

PHP 7 offered a major improvement over PHP 5 in terms of speed - up to twice the speed in most use cases. There are also many deprecations of old extensions and modules as well as a few new syntactic sugar operators (my favorite is the null coalescence operator). A bug in the OpenSSL extension for PHP prevented `fwrite()`s to non-blocking SSL sockets from completing. It was not found for so long due to the infrequency of non-blocking streams in PHP. Versions later than 7.1.4 include the patch for this bug.

I run the bot from my personal Ubuntu server. I run my own instance of GitLab and host close to a dozen websites with Apache. I learned how to problem solve the many different errors and bugs encountered when working with a headless server. Another obstacle I overcame with the hosting was the migration from my old server to this current machine. I built a new PC in the beginning of the year and decided to use my old desktop as the server instead of the laptop that was acting as the server at the time. I would gain CPU cores and double the RAM. It was an interesting challenge migrating all of my installed applications, sites, and configurations over to a new machine.

Composer is the de-facto standard for PHP package management. As any package manager should, Composer is able to install packages from `packagist.org` to the current project, writing exact version information to the `composer.lock` file. Other requirements can be specified in the `composer.json` file. I have leveraged the PSR-4 class autoloading feature provided by Composer to specify namespaces for my PHP classes.

## Source organization

The source code for my bot is organized into classes under the `src/` directory. Classes that contain code that will not be called as commands in Discord are in the root directory of `src/`. Classes that contain commands are in `src/Commands/`.

The `Command` class describes a command that the bot will perform for certain inputs. It holds the name of the command as a string, which is compared to the message when it

begins with the prefix `;`. If a match is found, the callable of the command is called with the specified parameters. With the current structure, classes in `src/Commands/` must be registered in the main `BenBot` class to get the BenBot instance as a class variable. The `register()` method in each of the Command classes then registers commands as needed to the BenBot instance.

The first iteration of my bot was a single file over 1000 lines long. This was unmanageable and difficult to work on. The only way to find a certain command or function was with the find function of my editor. The current structure is much easier to work with and I am able to add Command classes quickly and efficiently.

## The Hardest Part

I have continually found that the poor quality of documentation for DiscordPHP and lack of community around it has been the most difficult part of this project. I have spent many hours reading through the source code of DiscordPHP to understand the implementation details.

Other than the difficulty of finding solutions to my problems, it has also been difficult to wrap my brain around asynchronous code, especially the Promises paradigm. It finally began to click once I realized how I could interact with the results of promises and deal with exceptions (as rejected promises). When something failed or acted unexpectedly during the first few weeks of development, I didn't see any error messages. I thought it was due to PHP's strange handling of error messages, but it turned out that I needed to attach an `onRejected` promise handler to the methods I was using.

Now that I have a better handle on these issues, I have been able to move forward with the bot.

## Complex Data Structures and/or Algorithms

I'm rather proud of my implementation of TicTacToe. Players enter a number 1-9 to make moves, referring to each cell of the TicTacToe grid.

```
private static function getPieceAt($i)
{
    return $board[intval(($i - 1) / 3)][($i - 1) % 3];
}
```

Code Sample 1: Get TicTacToe cell value

```
if (self::placePieceAt($move, $player)) {
    if (self::checkWin()) {
        self::$bot->game['active'] = false;
        return "<@" . self::$bot->game['players'][self::$bot->game['turn']] . ">
    won";
    } else {
        self::$bot->game['turn'] = self::$bot->game['turn'] == ":x:" ? ":o:" :
    ":x:";
        return self::printBoard() . "\n<@" .
    self::$bot->game['players'][self::$bot->game['turn']] . ">, it's your turn!";
    }
} else {
    return "position already occupied!";
}
```

Code Sample 2: Handle a move from a player

Working with the OpenWeatherMap and Geonames APIs was also complicated. I opted to store the city, timezone, and coordinates for a city once it's been found to limit the number of API calls made.

```
public static function weather($msg, $args)
{
    $id = Utils::getUserIDFromMsg($msg);
    $api_key = getenv('OWM_API_KEY');
    $url =
    "http://api.openweathermap.org/data/2.5/weather?APPID=$api_key&units=metric&";

    if (count($args) <= 1 && $args[0] == "") {
        echo "looking up weather for {$msg->author} $id";
        if (isset(self::$bot->cities[$id])) {
            $url .= "id=" . self::$bot->cities[$id]["id"];
            self::$bot->http->get($url)->then(function ($result) use ($msg, $city) {
                Utils::send($msg, "", self::formatWeatherJson($result,
    $city["timezone"]));
            });
        } else {
            return "you can set your city with `;weather save <city>`";
        }
```

```
    } else {
        if (count($msg->mentions) > 0) {
            foreach ($msg->mentions as $mention) {
                if (isset(self::$bot->cities[$mention->id])) {
                    $url .= "id=" . self::$bot->cities[$mention->id]["id"];
                    self::$bot->http->get($url)->then(function ($result) use ($msg,
    $city) {
                        Utils::send($msg, "", self::formatWeatherJson($result,
    $city["timezone"]));
                    });
                } else {
                    return "no city saved for $mention.\nset a preferred city with
    `;weather save <city> $mention`";
                }
            }
        } else {
            $query = rawurlencode(implode(" ", $args));
            $url .= "q=$query";
            self::$bot->http->get($url)->then(function ($result) use ($msg) {
                Utils::send($msg, "", self::formatWeatherJson($result));
            });
        }
    }
}
```

Code Sample 3: Weather look up

## Expected Difficulty

This project has been a fun challenge. I have enjoyed working on it immensely. It has been an interesting mix of both easy and difficult. One one hand, it feels easy because of my familiarity with PHP, but on the other hand, the features that I planned on implementing in my Project Proposal turned out to be much harder than I anticipated.

The features that turned out being the hardest were:

- Audio Streaming to voice channels
- Games in chat

- Returning images from a web search

As I have discussed previously, these features were difficult due to the lack of documentation and initially brittle architecture of my bot. Additionally, my bot is best suited for working with REST-ful JSON APIs. Bing and Google have no easily-accessible image search API, so I would be unable to use the existing JSON architecture. To accomplish this, I will have to write an HTML parser to get the URL of the first image result.

Hopefully I will be able to build these features within the next week.

# Grade

This is the grading scale that I included with my Project Proposal.

Items in **boldface** text are incomplete and items in *italic* text are considered work-in-progress as of April 25, 2017.

- Get User Info [1 pts]
- Get Profile Photo for arbitrary User [1 pts]
- Get Server Info [2 pts]
- Display Bot Status/Uptime [1 pts]
- Send direct message to any user [1 pts]
- **Permissions for commands based on user's permissions [2 pts]**
- Talk to Cleverbot [3 pts]
- Save images and retrieve them later [5 pts]
- Save text and retrieve it later [2 pts]
- Send me a text message [2 pts]
- Send emails to a saved address for a user [2 pts]
- Internet lookups
  - Weather for any city [3 pts]
  - Time for any city (by looking up timezone) and formatting correctly [3 pts]
  - Save a preferred city for each user for time and weather [2 pts]
  - Look up a random joke [2 pts]
  - **Send an image from Google Image Search Results [5 pts]**

  - *Stream music from YouTube to a voice channel [10 pts]*

- Create and vote on polls [5 pts]
- Chat games (*TicTacToe*, **Hangman**, or both) [10 pts]
- Text transform (block emojis, unicode fonts, and ASCII art) [3 pts]
- Roll an n-sided die [1 pts]
- 8-Ball style fortunes [2 pts]

**70 points total.**
63+ $\rightarrow$ A
56+ $\rightarrow$ B
49+ $\rightarrow$ C
42+ $\rightarrow$ D
35+ $\rightarrow$ F

At the time of writing, I have completed 51 of 70 possible points, which falls in the C range. I hope to finish as many of the outstanding points as possible between now and my presentation.

# Code Samples


1 Get TicTacToe cell value . . . . . 4
2 Handle a move from a player . . . . . 5
3 Weather look up . . . . . 5