Instructions - Hello World Discord Bot

Introduction

This instruction will guide you through the steps needed to create your very first Discord bot!

Pre-Requisites

- Using `Windows 11` Operating System and is comfortable with basic functionality with it (Files Navigation, system settings, etc.)
- Have your own IDE installed and configured (recommended VSCode, which can be installed: https://code.visualstudio.com/Download)
- Have NodeJS installed (visit: https://nodejs.org/en/download for installation)

JavaScript (JS)

For this project, we'll be using JavaScript as our choice of language to write the bot.

Variable initialization

There are three ways to initialize a variable: `const`, `let`, and `var`. Use `const` if a defined variable will never change its value, and `let` if you do need to change.
NOTE: Avoid `var` unless absolutely necessary

Data Types

Here are some of the common data types to know:

- `number` - When a variable is assigned a numerical value (ex: `const a = 1.12`)
- `string` - When a variable is assigned some value inside quotation marks (ex: `const b = "STRING"`
- `boolean` - When a variable is assigned `true` or `false` value (`const c = true`)
- `undefined` - When a variable is not defined. This is different from `NULL`

Setup Hello World Project

Due to instruction limitations, we'll create a Discord bot that'll respond "pong" when executing a "ping" command. Discord Slash Command is a new way for users to run bot commands and here's how we'll set up the command.

1. Install Dependency

First, we'll install all the necessary dependencies. A dependency is another way to describe a project that you'll be integrating into your project.

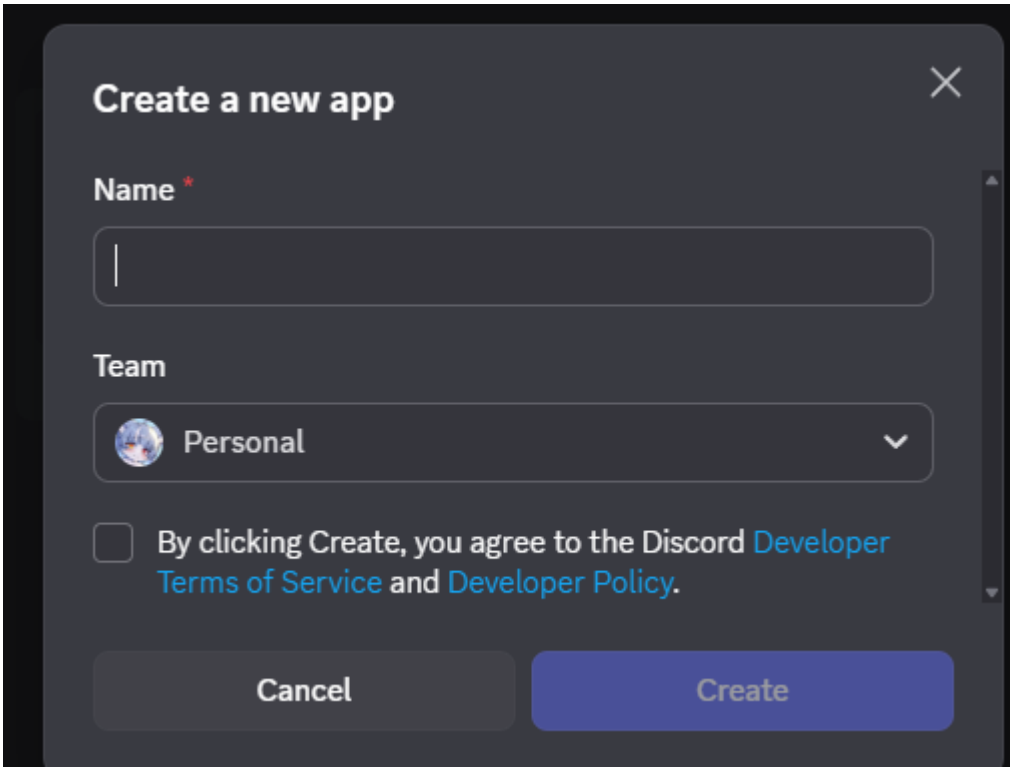
To install:

1. Initialize package.json by running `npm init` (you can just skip through all the process until the end)
2. Run `npm install discord.js` (for advanced users using other pkg manager, see https://discord.js.org/docs/packages/discord.js/14.25.1)

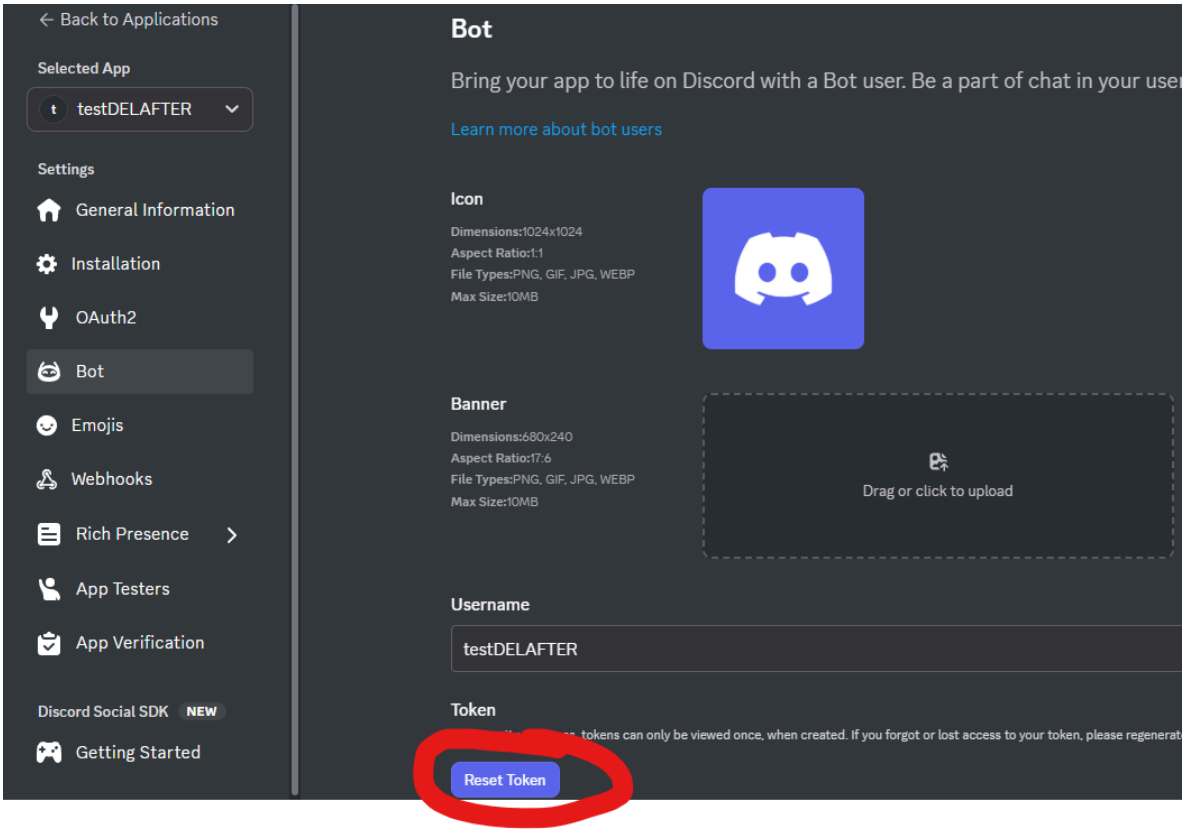
2. Discord Bot Token

Our next step is to create a bot profile and get the authentication token. You can do that by

1. Visit the Discord application page (make sure you're logged in first): https://discord.com/developers/applications
2. Click **New Application**, pick the name you want, agree to the Terms and Service, and click **Create**



3. On the **Settings** sidebar, click **Bot**
4. Customize the bot profile however you want (including names, icon, banner, etc.) For the project's purpose, we do not need any Privileged Intent.
5. Under Token, click **Reset Token** and save the token as a variable inside `index.js` (something like `const APIKey = "YOUR_TOKEN_HERE"`)
    - **NOTICE:** For project purpose, we will hardcode the API Key into a variable, but for production project, use an environment variable to store it!
        - An environment variable is a key/value that is stored in your system environment so that the value can be changed depending on which system the code is running on, without changing the source code
        - **Security Advisory:** Your API MUST remain secret at all times; leaking it will allow illicit actors to misuse your key and possibly cause your account to be terminated!

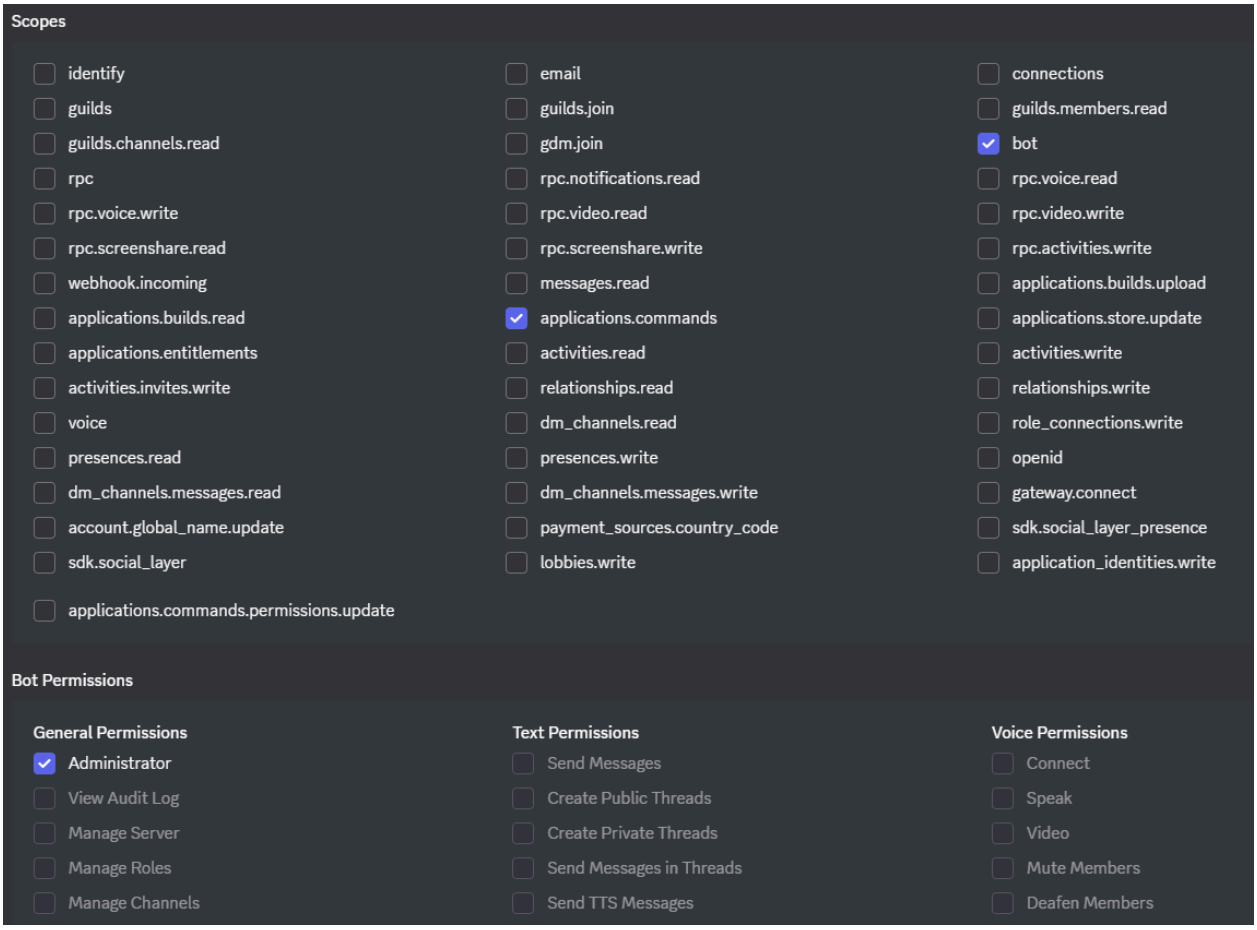


3. Invite bot to guild

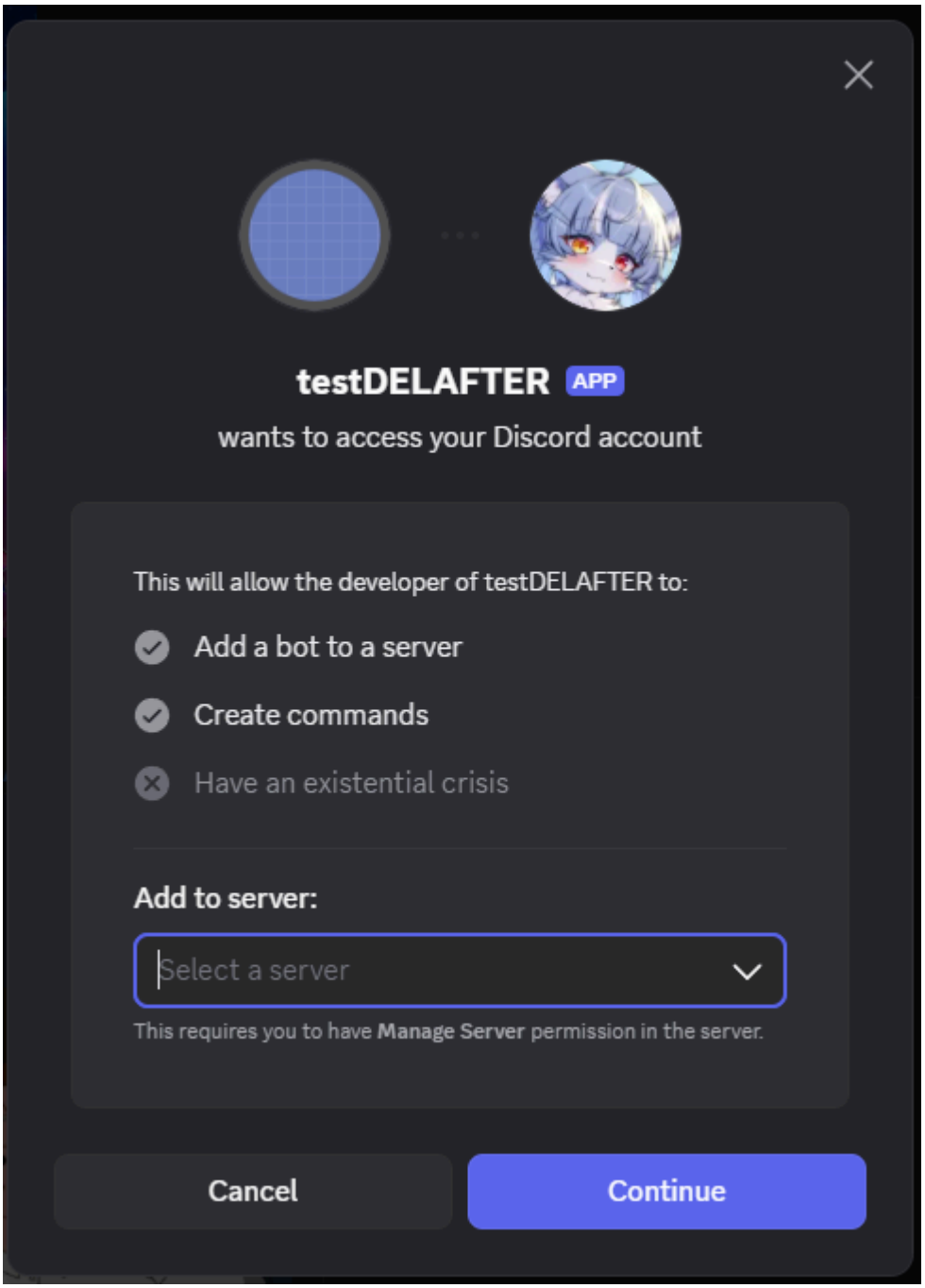
In order for you to test the bot, you'll have to invite it to your test server! To do so

1. Go to **OAuth2** on the sidebar and go to **OAuth2 URL Generator** section
2. Select `bot` and `applications.commands` permission and scroll down
3. For **Bot Permissions**, select `Administrator`
    - **Security Advisory:** In a production environment, you SHOULD NOT grant any Discord bot a full-administrator permission, as this allows the bot to potentially perform malicious

actions. Only grant specific permission as needed.



4. On the Integration Type, it should be **Guild Install**
5. Copy the generated URL and paste it into the browser
6. A window will pop up, asking you to select a server you want to add your bot to and click **Continue**



7. Once you finish it, you'll see the bot in the guild

Start Writing Codes!

Now that all the setups are completed, we can start writing code!

Publish Slash Command

First, we need to update all the available slash commands for the bot. This is not important for the project, but due to caching, you should only run this command when you update commands.

Here's the sample code to update the command:

```
const APIKey = "YOUR_API_TOKEN_HERE"
const { Routes, REST, SlashCommandBuilder } = require("discord.js");

// This is a helper class that allows you to define the command parameters
const command = new SlashCommandBuilder()
  .setName("ping")
  .setDescription("Respond with pong");

// This is the class that will be called to update your bot with the new sets
of commands
new REST({ version: "10" }).setToken(APIKey)
  .put(
    Routes.applicationCommands("Client ID"),
    { body: [command.toJSON()] }
  ).then(k=>console.log("Command Registered!"));
```

Make sure to replace "Client ID" with your Discord bot client ID, which can be found on the Discord application page.

Run the Bot

This code will set up event listeners and connect your bot:

```
const { Client, Events, GatewayIntentBits } = require("discord.js")

// This initializes the client class
const client = new Client({ intents: [GatewayIntentBits.Guilds] });

// This event is fired when the bot is ready
client.on(Events.ClientReady, readyClient => {
  console.log(`Logged in as ${readyClient.user.tag}!`);
});

// This event is fired when the user initiates any bot interactions
client.on(Events.InteractionCreate, async interaction => {
  if (!interaction.isChatInputCommand()) return;
  // Always use triple-equal sign; using double may lead to unexpected
```

```
behaviors
  if (interaction.commandName === 'ping') {
    await interaction.reply('Pong!');
  }
});

// This actually authenticates with the Discord server and connects your bot
to their platform
client.login(APIKey);
```

When you run the code, the bot should be started, and you can test your bot by running the command in your guild!

Sync/Async

In JavaScript, we can run code in an asynchronous manner (see async/await and .then()). The reason is a bit more complicated, but this allows multiple pieces of code to be run at the same time while it's waiting for a resource.

Event listeners

Listeners or eventEmitter is a way for JavaScript code to fire multiple callbacks that are listening to an event. A callback is when a function calls another function when the original function finishes execution

---

Conclusion

Congratulations on your first bot! Of course, this is only a very basic level of the bot, and hopefully, you'll be able to learn further to improve the functionality of the bot.