



Discord API Getting Started Guide

TABLE OF CONTENTS

| | |
|--|----|
| Welcome to the Discord API..... | 2 |
| Getting Set-Up..... | 2 |
| Step 1: Activating your Bot | 2 |
| Step 2: Bot meet Server, Server meet Bot..... | 5 |
| Step 3: Making your Bot Work | 5 |
| <i>Setting up your laboratory! (Environment)</i> | 6 |
| <i>Discord.js (Ping-Pong)</i> | 7 |
| <i>Discord.js (d20 System)</i> | 7 |
| <i>Discord.py (Ping-Pong)</i> | 8 |
| <i>Discord.go (Ping-Pong)</i> | 9 |
| Step 4: Running your Bot | 10 |
| Conclusion..... | 10 |
| Links to Official Bots | 10 |

WELCOME TO THE DISCORD API

Welcome future content contributor to the Discord API! This document is meant to introduce anyone new to our API, or the veterans that get bogged down by coder's block. In the references below you'll find explanations of the functions of our API, code examples of our API in use, and highlighted links to our Documentation pages so you can see how we do it here at Discord.

This document is going to walk you through the creation of your own bot and some basic examples. At the end of the document take a look at our official [Airhorn Bot](#). Airhorn Bot is programmed in Go, which this document has an example of using with a "ping-pong" bot.

Links for each language tie-in mentioned can be found [below](#).

GETTING SET-UP

You've probably been on a Discord server that may have a simple command to summon the thundering sound of an air horn, a robot DJ playing the latest music, or a webhook telling you, and anyone in the channel, all those commits you just made to GitHub. You're here because you want to make something neat, or maybe you've come back to check out what's new and shiny.

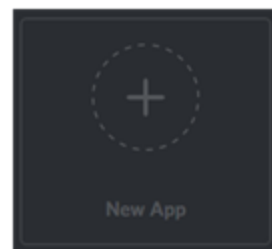
With that said, let's get started with our friend, exampleBot!

STEP 1: ACTIVATING YOUR BOT

This is the first step for all applications that tie into Discord.

First, you'll need to [register](#) an account with or [log in](#) to Discord!

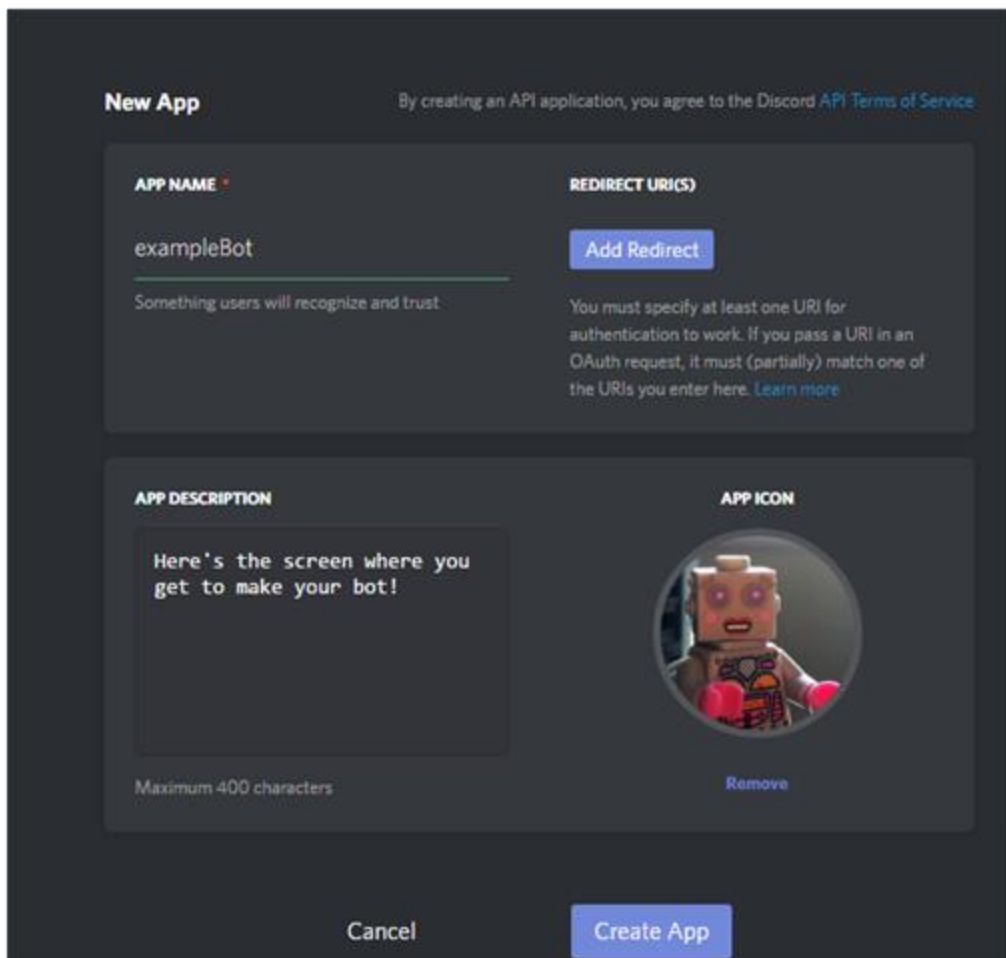
Now you're going to want to go to the [Discord Developer Page](#)



Click on the button that says **New App**.

Clicking on this will bring you to the **New App** page.

Here, you just need to enter an **App Name** of your new application or bot. For right now we're going to make exampleBot to lead us through this journey!

The image shows the 'New App' creation interface in Discord. At the top, it says 'New App' and 'By creating an API application, you agree to the Discord API Terms of Service'. The form is divided into two main sections. The top section has two columns: 'APP NAME' and 'REDIRECT URI(S)'. Under 'APP NAME', there is a text input field containing 'exampleBot' and a subtext 'Something users will recognize and trust'. Under 'REDIRECT URI(S)', there is a blue 'Add Redirect' button and a note: 'You must specify at least one URI for authentication to work. If you pass a URI in an OAuth request, it must (partially) match one of the URIs you enter here. Learn more'. The bottom section has two columns: 'APP DESCRIPTION' and 'APP ICON'. Under 'APP DESCRIPTION', there is a text area containing 'Here's the screen where you get to make your bot!' and a subtext 'Maximum 400 characters'. Under 'APP ICON', there is a circular image of a LEGO minifigure and a blue 'Remove' button. At the bottom of the form, there are two buttons: 'Cancel' and 'Create App'.

Next to the App Name, you'll see a button for **Redirect URI(s)** (Uniform Resource Identifier)

This is optional, just keep in mind that for OAuth to work, at least one valid URI must be used.

A brief **App Description** is also helpful in reminding you what this bot does in your future bot army!


The **Application Icon** off to the side also lets you keep track of your bot with a unique picture.


After this is finished, **click on Create App** to move to the next page.

REVIEWING: **exampleBot** By creating an API application, you agree to the Discord [API Terms of Service](#)

GREAT SUCCESS!
Your sweet new application has been created successfully!


APP DETAILS
Client ID: 279008899628007424
Client Secret: [click to reveal](#)

 You can bundle a Bot User with your app to interact with users in a more conversational manner. **This action is irreversible.** [Learn more about bot users](#) [Create a Bot User](#)

APP NAME 
exampleBot
Something users will recognize and trust

REDIRECT URI(S)
[Add Redirect](#)
You must specify at least one URI for authentication to work. If you pass a URI in an OAuth request, it must (partially) match one of the URIs you enter here. [Learn more](#)

APP DESCRIPTION
Testing out Discord Bot Functionality
Maximum 400 characters

APP ICON

[Remove](#)

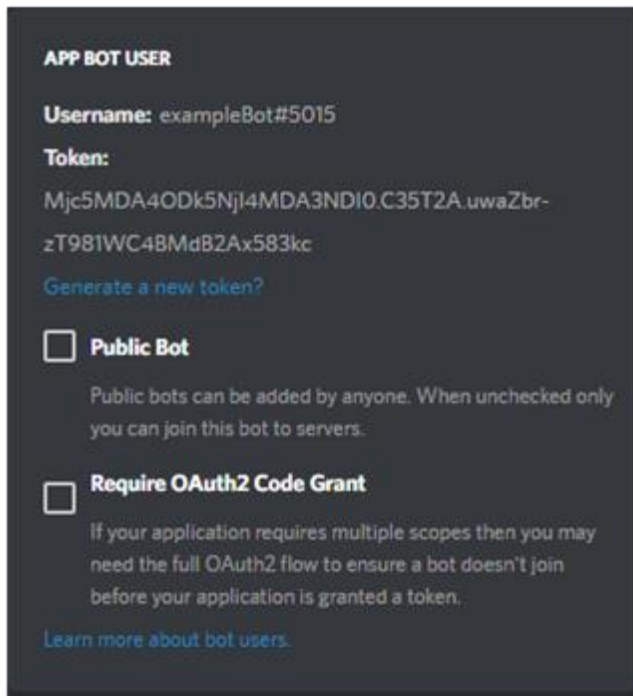
Your application or bot is now created! Hooray!

Here you can review and change items such as **App Name**, **App Description** and the **App Icon**. If a URI are needed you can also add in a new **Redirect URI**.

New Information found in the App Details Panel can be viewed as well. This is your **Client ID** and the **Client Secret**.

WARNING: When it comes to your **Client Secret**, **do not share this**. The **Client Secret** acts as your login to Discord and all your applications which are tied to it.

We can also **Create a Bot User** here, which for the purposes of making exampleBot live up to her name, we're going to click on it.



APP BOT USER

Username: exampleBot#5015

Token:
Mjc5MDA4ODk5Njl4MDA3NDI0.C35T2A.uwaZbr-
zT981WC48Md82Ax583kc

[Generate a new token?](#)

☐ **Public Bot**
Public bots can be added by anyone. When unchecked only you can join this bot to servers.

☐ **Require OAuth2 Code Grant**
If your application requires multiple scopes then you may need the full OAuth2 flow to ensure a bot doesn't join before your application is granted a token.

[Learn more about bot users.](#)

Clicking on **Create a Bot User** changes our application screen to include this new information.

exampleBot now has a **Username**, complete with hashtag and a string of numbers, that is like an ordinary username with Discord.

WARNING: Because the **Token** is how exampleBot logs into any application, please **do not share this**. This Token is similar to the **Client Secret** found above and exampleBot is not a fan of having her unique identity stolen by anyone.

We can also make exampleBot a **Public Bot** so that anyone can add her to their server. There is an option to also **Require OAuth2 Code Grant**. This is important to make sure that exampleBot cannot join a server without getting a **Token** in return.

STEP 2: BOT MEET SERVER, SERVER MEET BOT

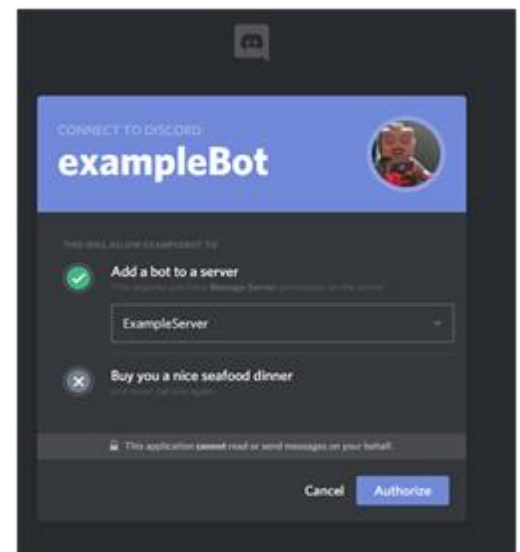
Our exampleBot needs a server to call home now! That **ClientID** is about to become useful for this next section.

Found in [this section](#) of the Discord API, we are now going to add the bot to our channel. To add it, we use the following line in your browser:

```
https://discordapp.com/api/oauth2/authorize?client_id=[ClientID here]&scope=bot&permissions=0
```

Which will cause this connection page to pop-up. From the dropdown you'll want to select the server you'd like this application to be on and **Authorize** it!

Great! It looks like exampleBot now has a home on ExampleServer!



STEP 3: MAKING YOUR BOT WORK

Now for the fun part, we're going to give exampleBot a function! Now that we've authorized her with the Discord API we're moving to the next fun step, which is going to require a few tools before we start.

Disclaimer: if you're new to programming, we recommend checking out some introduction material on the language that you find the most interesting, before launching into bot programming. The examples here are also rather simple, links will be provided [below](#) for bigger projects and examples.

Just below you'll find links to several instruction sets on setting up your own application/bot laboratory, also known as an environment! You will also find the Discord API tie-ins that content creators have already made for JavaScript, Python and Go.

| Languages | JavaScript | Python | Go |
|---------------|----------------------------|----------------------------|----------------------------|
| Main Files | Node.js | Python 2.x | Go |
| Documentation | Discord.js | Discord.py | Discord.go |

Now to make something! Below you'll see a pseudocode example, as well as the three languages mentioned above of how to create a "Ping-Pong Bot" which takes in a message, "Ping" and responds with "Pong". This will show you the basics of getting your application to talk to a Discord server as well as send/receive functionality.

This document will only break down how to create and run the JavaScript version of this example.

SETTING UP YOUR LABORATORY! (ENVIRONMENT)

Whether you're on Mac/Linux/Windows we recommend [Atom](#), which is a text editor made by the [GitHub](#) team. Speaking of GitHub, it is an excellent idea to make an account with them and get familiar with version control. This will let you share all your bot creations with the world!

At this point we recommend making a separate folder within your system where you can easily locate all your bot projects. For example, we're going to make this bot right in our botArmy folder using terminal:

```
mkdir /Users/exampleUser/botArmy/exampleBot
cd exampleBot/
```

Depending on the language you've selected, it's also a good time to pull down the files for that language and make sure they are ready to use. For Discord.js you can type in the following command in your command line:

```
npm install --save discord.js
```

Next stop, let's open our text editor and make an exampleBot.js file. When you're done, you can move to [Step 4](#) if you want to skip past the other example of languages.

DISCORD.JS (PING-PONG)

```
var Discord = require('discord.js'); //Acts as the import for discord.js
var exampleBot = new Discord.Client(); //Creating a bot that acts as a client
var botToken = 'Your Token Here'; //This is the bot's ClientID token

exampleBot.on('message', botMsg => { //We're making a string message and capturing it
  if (botMsg.content.startsWith('ping')) { //If the user types in 'ping' we move forward
    botMsg.channel.sendMessage('pong!'); //Bot sends a message back, 'pong!' in this case
    console.log('pong-ed ' + botMsg.author.username) //Error check on console side
  }
});
exampleBot.login(botToken); //Authenticates the bot
```

DISCORD.JS (D20 SYSTEM)

```
var Discord = require('discord.js');
var rollerBot = new Discord.Client();
var rollerToken = 'Mjc5MDY3Njc1OTk1NjY4NDgw.C31djA.dkUaf8thdp-h5IAIKouHfgEL2Y';
var diceResult = 0;
//Rollbot, tell them you're ready!
rollerBot.on('ready', message => {
  message.channel.sendMessage('Ready to roll!');
});

//The user specifies what dice they'd like rolled and rollerBot makes a random number!
rollerBot.on('message', message => {
  if (message.content === 'd20'){
    diceResult = 1+Math.floor(Math.random() * (20+1)); //The +1 occurs because standard dice do not have 0's on them
    message.channel.sendMessage('You rolled a ' + diceResult);
  }
  else if (message.content === 'd12'){
    diceResult = 1+Math.floor(Math.random() * (12+1));
    message.channel.sendMessage('You rolled a ' + diceResult);
  }
  else if (message.content === 'd10'){
    diceResult = Math.floor(Math.random() * (10+1)); //D10 is unique as dice sets use 0 in order to do percentages
    message.channel.sendMessage('You rolled a ' + diceResult);
  }
  else if (message.content === 'd8'){
    diceResult = 1+Math.floor(Math.random() * (8+1));
    message.channel.sendMessage('You rolled a ' + diceResult);
  }
  else if (message.content === 'd6'){
    diceResult = 1+Math.floor(Math.random() * (6+1));
    message.channel.sendMessage('You rolled a ' + diceResult);
  }
  else if (message.content === 'd4'){
    diceResult = 1+Math.floor(Math.random() * (4+1));
    message.channel.sendMessage('You rolled a ' + diceResult);
  }
});
rollerBot.login(rollerToken);
```

DISCORD.PY₁ (PING-PONG)

```
from disco.bot import Bot, Plugin

class SimplePlugin(Plugin):
    # Plugins provide an easy interface for listening to Discord events
    @Plugin.listen('ChannelCreate')
    def on_channel_create(self, event):
        event.channel.send_message('Woah, a new channel huh!')

    # They also provide an easy-to-use command component
    @Plugin.command('ping')
    def on_ping_command(self, event):
        event.msg.reply('Pong!')

    # Which includes command argument parsing
    @Plugin.command('echo', '<content:str...>')
    def on_echo_command(self, event, content):
        event.msg.reply(content)
```

¹ Thank you [b1naryth1ef](#)

DISCORD.GO₂ (PING-PONG)

```
package main
import (
    "flag"
    "fmt"
    "github.com/bwmarrin/discordgo"
)
// Variables used for command line parameters
var (
    Token string
    BotID string
)
func init() {
    flag.StringVar(&Token, "t", "", "Bot Token")
    flag.Parse()
}

func main() {
    // Create a new Discord session using the provided bot token.
    dg, err := discordgo.New("Bot " + Token)
    if err != nil {
        fmt.Println("error creating Discord session,", err)
        return
    }
    // Get the account information.
    u, err := dg.User("@me")
    if err != nil {
        fmt.Println("error obtaining account details,", err)
    }
    // Store the account ID for later use.
    BotID = u.ID

    // Register messageCreate as a callback for the messageCreate events.
    dg.AddHandler(messageCreate)

    // Open the websocket and begin listening.
    err = dg.Open()
    if err != nil {
        fmt.Println("error opening connection,", err)
        return
    }
    <-make(chan struct{})
    return
}
// This function will be called (due to AddHandler above) every time a new
// message is created on any channel that the authenticated bot has access to.
func messageCreate(s *discordgo.Session, m *discordgo.MessageCreate) {

    // Ignore all messages created by the bot itself
    if m.Author.ID == BotID {
        return
    }

    // If the message is "ping" reply with "Pong!"
    if m.Content == "ping" {
        _, _ = s.ChannelMessageSend(m.ChannelID, "Pong!")
    }

    // If the message is "pong" reply with "Ping!"
    if m.Content == "pong" {
        _, _ = s.ChannelMessageSend(m.ChannelID, "Ping!")
    }
}
```

2 Thank you [bwmarrin](#)

STEP 4: RUNNING YOUR BOT

For this step, we're going to take exampleBot and see how she responds to a ping command.

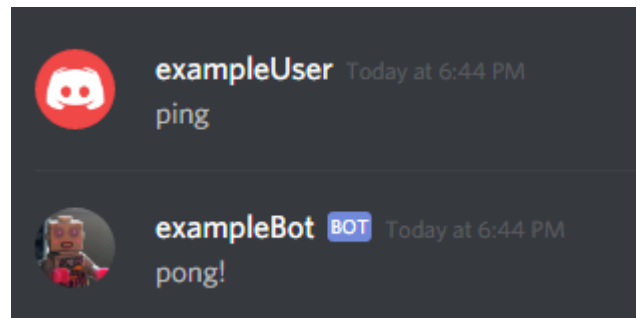
First, we're going to have to tell exampleBot to start listening for a ping. Then, we'll navigate to where our exampleBot.js file is and run the following command:

```
node exampleBot.js
```

This will start exampleBot up and when we type in our ping command she should respond with a "pong!".

Look at that! Seems like exampleBot is already taking in a response and sending out a message to everyone on the server.

Congratulations on taking your first steps in bot creation with the Discord API!



CONCLUSION

This concludes the Discord API Getting Started Guide. We here at Discord hope that with the information provided you'll make a Bot army of your own! See the links [below](#) to check out other fun Bots created with the Discord API and the tools found [above](#).

LINKS TO OFFICAL BOTS

| Bot | Function |
|-----------------------------|---|
| Airhorn Bot | Blasts an Airhorn when !Airhorn is typed. Written in Go |